Departamento de Computación
Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN INGENIERÍA DEL SOFTWARE

Sistema para la gestión de méritos del PDI y la generación automática de currículums

Estudiante: Borja Álvarez García
Director/a/es/as: Miguel Ángel Rodríguez Luaces
Alejandro Cortiñas Álvarez

A Coruña, 5 de septiembre de 2019.

A mi abuelo Guillermo, por haber estado siempre ahí.

Agradecimientos

A mi familia, por haberme guiado hasta aquí y a mis directores, por haber hecho posible este proyecto.

Resumen

Este proyecto se basa en el análisis, diseño e implementación de un sistema que permita al usuario la gestión de su currículum de manera eficiente y la generación de diferentes instancias del mismo. El modelo con el que se generarán estas instancias será elegido por el usuario cada vez que desee crear un currículum, dentro de los diferentes plantillas ofrecidas por la aplicación. Estas plantillas no solo se diferencian estéticamente sino que cada una se ajustará a un formato de currículum específico. Esto permitirá que el usuario, a través del mismo conjunto de méritos introducidos una única vez en la aplicación, cree un currículum simple o uno que siga formato complejos y específicos como puede ser el currículum para la solicitud de plazas en la UDC. En el desarrollo del proyecto se ha optado por la creación una arquitectura cliente-servidor, empleando para ello tecnologías como Spring o React. Para el almacenamiento de la información se ha empleado un SGBD como PostgreSQL y para el control de versiones se ha utilizado GitLab. El trabajo ha sido gestionado siguiendo una metodología fruto de la adaptación de las mejores prácticas de SCRUM a las peculiaridades del proyecto.

Abstract

This project is based on the analysis, design and implementation of a system that allows the user the management of his curriculum in an efficient way and the generation of different instances of it. The model that will generate those instances will be chosen by the user every time he wants to create a curriculum, within the different templates provided by the application. These templates do not present just aesthetic differences but they would adjust each one to a format of a specific curriculum. This will permit the user, using the same data, the possibility of creating a simple curriculum or more complex formats like the place solicitude curriculum from UDC. In the project development, the choice made was the creation of a client-server architecture, coded in React and Java respectively. The information storage was accomplished using PostgreSQL and the version-control with the help of GitLab. The work was managed using as methodology an adaptation of the SCRUM's best practices to the peculiarities of the project.

Palabras chave:

- Curriculum
- Plantillas
- Gestión

Keywords:

- Curriculum
- Templates
- Management

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1 Motivación

La creación de currículums siempre es un proceso laborioso, largo y repetitivo. Esta problemática es debida a las diferencias entre los formatos de currículums existentes, que muchas veces, a pesar de que solo difieren en la manera de estructurar la información, obligan al creador a reintroducir todos sus datos reiterativamente cada vez que desea crear un nuevo tipo de currículum. Además, el usuario no solo pierde tiempo introduciendo los datos de nuevo, sino que también acaba perdiendo mucho tiempo en el formateado, diseño y estructuración del currículum a través de las plantillas de texto que debe usar.

Debido a esta situación, surge la necesidad de disponer de una aplicación que no solo facilite la gestión de los datos y la creación de currículums, sino que evite tener que introducir de nuevo todos los datos cada vez que se quiera crear un formato de currículum diferente. Además, también debe abstraer al usuario del diseño de las plantillas, permitiéndole únicamente decidir que información quiere introducir en el currículum, y después obtener el resultado final sin preocuparse de ningún otro aspecto del mismo.

El objetivo del proyecto es la creación de una aplicación sencilla, que permita al usuario la introducción y gestión de sus datos de una manera cómoda, evitando además repetir esta introducción de datos cuando quiera generar una nueva versión de su curriculum. Sumado a esto, la aplicación debe ser flexible y parametrizable, para permitir que con los mismos datos se puedan crear tantos formatos de currículums como plantillas ofrezca la aplicación. Así, el usuario solo deberá introducir los méritos y seleccionar el formato que desee, y la aplicación se encargará de generar automáticamente el currículum. Por último, el sistema debe ser flexible para dar soporte a cualquier formato de currículum de forma sencilla.

1.2 Objetivos

Como hemos descrito anteriormente, el objetivo de este proyecto es la creación de una aplicación de gestión de méritos del usuario que permita la exportación de estos datos en currículums de diferentes formatos. Para dar soporte a esto, la aplicación se compone de las siguientes funcionalidades principales:

- Gestión de autenticación y registro de usuario.
- Gestión de datos personales y méritos laborales y académicos.
- Creación y exportación de currículums ajustados a diferentes formatos seleccionables.
- Transformación y adaptación de datos ya introducidos para que sean reutilizables en nuevos formatos de currículum.

El sistema debe ser completamente flexible, parametrizado y adaptable, lo que permite dar soporte a cualquier formato de currículum. Para esto, los méritos del usuario se crean asociados a una categoría o tipo de mérito, que determina la manera en la que se guarda la información. Estas categorías son completamente flexibles y modificables permitiendo que la aplicación pueda almacenar cualquier estructura de información deseada, sin existir límites de ningún tipo.

Además, los currículums están formados dinámicamente por secciones independientes, lo que permite que cada formato de currículum evolucione con el tiempo y que no esté nunca sujeto a una estructura fija o limitado por las categorías de datos existentes. La flexibilidad y simpleza para añadir nuevos formatos de currículum se consigue gracias a que todos los datos que se almacenen en la aplicación pueden ser transformados dinámicamente en cualquier momento, dando lugar a versiones adaptadas del mismo dato para el nuevo formato de currículum.

1.3 Estructura del trabajo

A lo largo de esta memoria, describiremos el desarrollo del proyecto a través de diferentes ámbitos y puntos de vista. A continuación, desglosaremos brevemente los capítulos que nos encontraremos y que trataremos en cada uno de ellos:

- En **Estado del arte**, conoceremos otras aplicaciones con funcionalidades similares a las de este proyecto. Esto nos permitirá ver similitudes y diferencias con otras soluciones ya existentes en el mercado y ver qué es lo que la aplicación que hemos desarrollado aporta, que no hacen las otras.

- En **Tecnologías**, introduciremos todas las herramientas que se han empleado en el desarrollo de la aplicación, profundizando en alguna especialmente relevante.
- En **Metodologías del desarrollo**, explicaremos la metodología usada en el desarrollo del proyecto. Para esto, veremos las características de la misma, cómo la hemos adaptado a las peculiaridades del proyecto y cómo divide el proceso del desarrollo.
- En **Planificación y Seguimiento**, veremos cuáles son las tareas del proyecto y los recursos que se han empleado para el desarrollo. Además, veremos como se ha distribuido el trabajo en el tiempo y su adecuación con la estimación del desarrollo.
- En **Análisis**, veremos cuales son los requisitos del proyecto, la arquitectura del sistema y los detalles de la interfaz de usuario y el modelo de datos. Ilustraremos lo anterior ayudándonos de diagramas y prototipos de pantallas.
- En **Diseño**, realizaremos una descripción tecnológica del sistema. Veremos cuáles son los componentes, cómo interactúan entre sí, así como los diagramas de la lógica de la aplicación.
- En **Implementación y pruebas**, analizaremos los detalles de la codificación de una manera más exhaustiva y profunda. Explicaremos detalles complejos de la implementación, así como las pruebas a las que hemos sometido al sistema a lo largo del desarrollo.
- En **Solución desarrollada**, mostraremos las características principales de la aplicación ya terminada, ilustrándolas con capturas de la aplicación. Este capítulo nos acercará a conocer cómo es visual y funcionalmente la aplicación ya terminada.
- Por último, en **Conclusiones**, analizaremos el grado de completitud que ha alcanzado el proyecto y su utilidad en el mundo real. A mayores, analizaremos algunas funcionalidades o modificaciones interesantes para el sistema, que no se han realizado por encontrarse fuera del espectro del proyecto.

Capítulo 2

Fundamentos tecnológicos

2.1 Estado del arte

2.1.1 Introducción

Es cierto que en el mercado ya existen diferentes herramientas de edición y creación de currículums que se adecúan a diferentes formatos preestablecidos. Sobre todo, existen un sinfín de aplicaciones de creación de currículums genéricos, pero el número se ve muy reducido cuando intentamos buscar editores de formatos más complejos y específicos, como son los necesarios para el personal de investigación.

Además, otro de los grandes problemas existentes actualmente es que estos editores son generalmente específicos de un único formato. Esto fuerza al usuario a tener que emplear una aplicación diferente para cada formato que necesite crear y a tener que reintroducir los numerosos datos que los forman tantas veces como formatos diferentes de currículum necesite. Dentro de estas herramientas nos encontramos aplicaciones para la creación de currículums genéricos como podría ser Europass [1] o editores de currículums más complejos y específicos como el editor de CVN de la FECYT [2].

2.1.2 Editor Europass

Europass es una aplicación web que permite la creación de currículums genéricos a través de una plantilla editable. Permite añadir cualquier tipo de información personal, méritos tanto profesionales como estudiantiles, así como adjuntar documentos a estos.

Las categorías y los datos son los preestablecidos por la aplicación que los estructura de una forma determinada. La flexibilidad que presenta es únicamente que el usuario puede decidir qué categorías desea incluir o excluir del currículum final. Sin embargo, el usuario no puede añadir categorías personalizadas ni seleccionar otro tipo de formato.

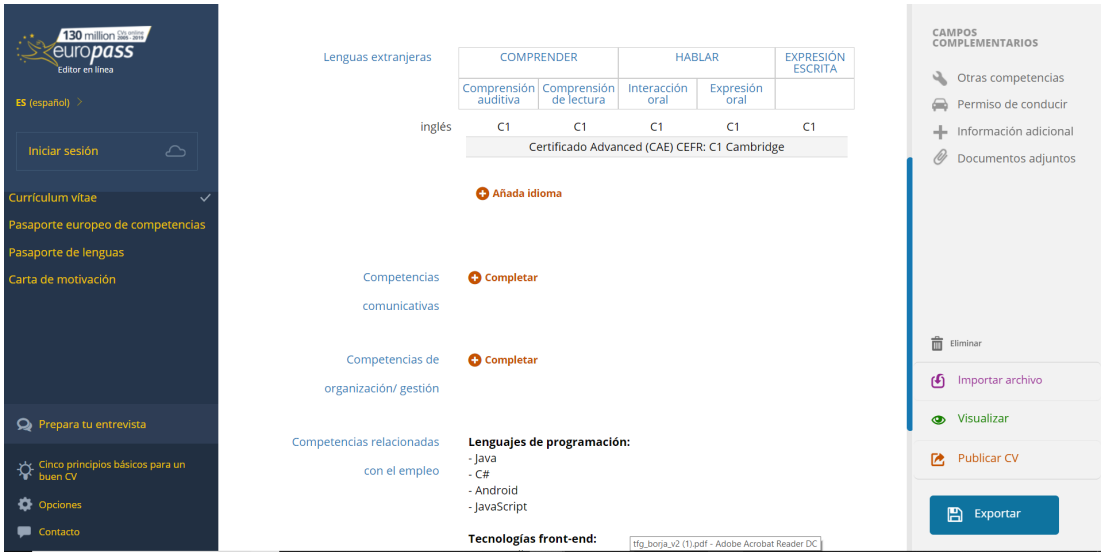

Figura 2.1: Editor de CVs de Europass

2.1.3 Editor de CVNs de la FECYT

La FECYT proporciona una aplicación de edición de currículums adecuada a la generación tanto de CVN como de CVA. La creación del currículum se realiza a través de la introducción de datos en los formularios existentes para cada categoría de las necesarias para completar el CVN. Una vez rellenadas todas las categorías deseadas, permite la exportación tanto de un CVN como de un CVA.

Como funcionalidad interesante destaca la posibilidad de importar tus datos a través de un currículum ya realizado previamente a través de la aplicación. Esto permite que la actualización de un currículum ya realizado no sea un proceso tan tedioso. Además también permite consultar un simple manual donde se indica la estructura que tendrá el currículum final.

2.1.4 Conclusiones

Como hemos visto, sí existen aplicaciones web de creación de currículums más o menos complejos, pero todas carecen, sobre todo, de la flexibilidad de poder transformar los datos del usuario a múltiples formatos diferentes y no solo a uno o dos únicamente.

El proyecto es similar a estas aplicaciones, en cuanto a que presenta un gestor y editor simple de datos organizados por categorías. Además, también presenta la posibilidad de adjuntar documentos, así como la consulta de manuales on-line para saber cómo se debe proceder para crear cada formato que exporte la aplicación.

Entre todas las funcionalidades diferentes que hacen que la aplicación pueda ser más atractiva para el usuario, destaca la flexibilidad de la aplicación. La aplicación puede generar, con

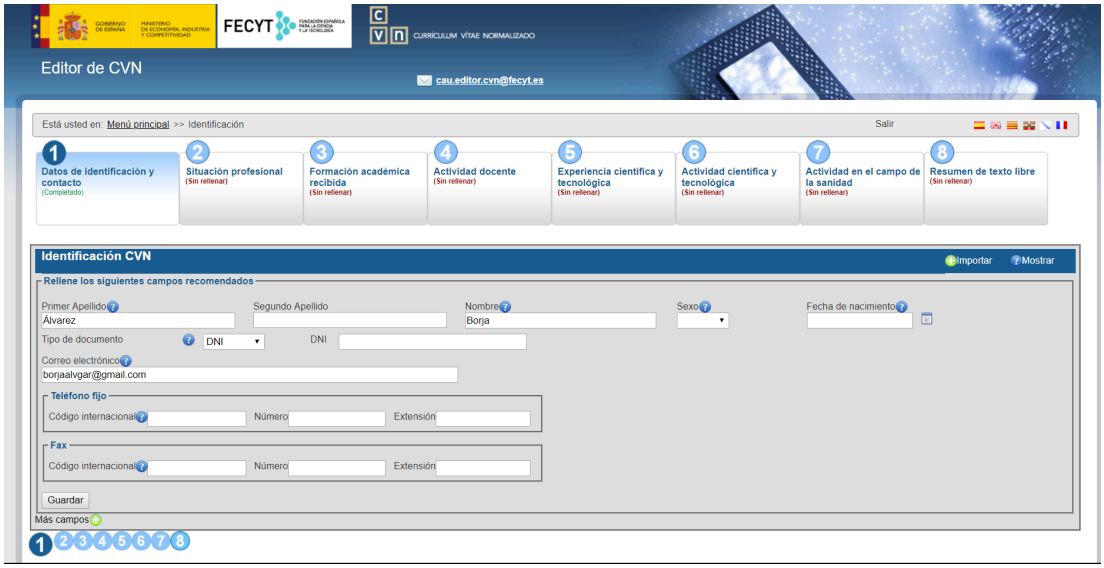

Figura 2.2: Editor de CVNs de la FECYT

los mismos datos, tantos currículums diferentes como plantillas haya disponibles en el sistema. Además, no existen límites en la generación de formatos, ya que para permitir al usuario generar un formato de currículum diferente únicamente se debe crear e incluir una plantilla nueva. Por último, también se permite la creación de funciones JavaScript personalizadas para permitir al usuario transformar los datos ya existentes, de la manera que se desee, para adecuarlos a nuevos formatos. Esto evita los problemas de las aplicaciones que hemos comentado anteriormente, como la imposibilidad de importar los datos para crear un formato diferente de currículum, ya que solo permiten importar los datos de un currículum del mismo formato para su actualización.

2.2 Tecnologías empleadas

2.2.1 Introducción

Para la construcción de este proyecto, se han empleado diferentes tecnologías en todas las etapas del desarrollo.

- Java: Lenguaje de programación orientado a objetos. [3]
- Spring Boot: Framework Java para el desarrollo aplicaciones. [4, 5]
- Apache PDFBox: Biblioteca Java para la edición de metadatos de archivos PDF. [6]
- JasperReports: Biblioteca de creación de informes. [7, 8]

- React: Biblioteca JavaScript de creación de interfaces de usuario. [9, 10]
- JavaScript: Lenguaje de programación interpretado. [11]
- Babel: Compilador e intérprete JavaScript. [12]
- Material-UI: Framework de componentes React. [13]
- MUICSS: Framework de componentes React. [14]
- JSONWebToken: Estándar abierto para la creación de tokens de acceso. [15, 16]
- Eclipse: Plataforma Software para el desarrollo de aplicaciones. [17]
- VisualStudio Code: Editor de código fuente. [18]
- Soap.ui: Herramienta para la realización de pruebas de APIs. [19]
- GitLab: Aplicación web de control de versiones y gestión del ciclo de vida del proyecto. [20]

2.2.2 JasperReports

De todas las tecnologías nombradas cabe destacar, dentro del funcionamiento de la aplicación, la librería JasperReports. Esta librería es uno de los grandes pilares sobre los que se edifica el proyecto, ya que permite la creación de todas las plantillas deseadas de currículums, así como su rellenado dinámico a través de los datos proporcionados por el usuario.

Para el desarrollo de las plantillas, hemos utilizado la aplicación de creación JasperSoft Studio [21], que simplifica la creación de los archivos XML que las constituyen. Para su compilación y posterior rellenado dinámico, nos hemos valido de la librería JasperReports en Java, que a su vez crea finalmente el archivo PDF resultante.

Capítulo 3

Metodologías del desarrollo

3.1 Introducción

Este proyecto se ha construido siguiendo una metodología que adapta las mejores prácticas de SCRUM [22] a las necesidades del desarrollo del sistema. SCRUM es una metodología de desarrollo ágil cuyo objetivo es obtener el mejor resultado y en el menor tiempo posible. SCRUM es una metodología basada en ciclos o sprints, generalmente cortos, de una duración determinada, donde cada sprint está formado por las mismas etapas que todos los demás y debe terminar con un sistema completo, presentable y analizable. Además, se destaca la necesidad de que el cliente o promotor de la aplicación (*product owner*) debe estar involucrado en el desarrollo del sistema y debe ser una parte principal en la toma de decisiones, ya que esto derivará en que la aplicación termine ajustándose mejor a sus expectativas.

Como característica principal de la metodología, podríamos destacar que está basada en el valor, es decir, las tareas se realizan en base a la prioridad que se les asigna al principio de cada sprint. Estas prioridades pueden ser modificadas antes de comenzar cada sprint, en función de cómo avancen las necesidades del cliente. Esta replanificación constante, sumada a ciclos cortos, es lo que hace que SCRUM sea una metodología que aporta al desarrollo una capacidad de adaptación y mejora, que no permitían metodologías más clásicas, como podrían ser la metodología en cascada o en espiral.

Como SCRUM es una metodología de trabajo para equipos de desarrollo, se ha adaptado para poder emplear las mejores prácticas de la misma aplicándolas a un equipo de desarrollo formado por un único desarrollador. Así, se han eliminado prácticas comunes como pueden ser las *daily meeting* o reuniones de actualización de trabajo diarias, al ser inoperativas en un desarrollo individual del sistema.

Cada uno de los *sprints* que conforman el desarrollo del sistema está compuesto de las siguientes fases:

Reunión inicial de planificación: Se trata de una reunión realizada antes de empezar ca-

da sprint, que permite seleccionar los requisitos a realizar durante el ciclo. Para esto, primero se debe crear un *product backlog* o lista de tareas que quedan por realizar para acabar el sistema. Todas las tareas que componen este *product backlog* serán priorizadas, obteniendo de este modo una lista de tareas ordenadas por el interés del cliente. Dentro de esta lista, seleccionaremos las tareas a realizar durante el sprint, basándonos en la estimación de tiempo de duración de cada una. Así, las tareas a realizar durante cada sprint, presentarán un esfuerzo estimado abarcable para el desarrollador.

Ejecución del Sprint: Etapa que se corresponde con la codificación y desarrollo de las tareas previamente planificadas. Durante esta etapa, simultáneamente con la codificación y desarrollo de pruebas del sistema, se registra el tiempo empleado por el desarrollador para compararlo al final de cada ciclo, y replanificar los futuros *sprints* de ser necesario.

Revisión y Retrospectiva: Una reunión al finalizar el ciclo, que permite evaluar el resultado final del ciclo y comprobar si se han alcanzado los requisitos planificados al inicio. Durante el mismo, se observará que el sistema obtenido es el que se esperaba y, de no ser así, se crearán tareas de refactorización y se añadirán al *product backlog*. Además, se comprobarán las diferencias entre los tiempos estimados y los inputados. Estos análisis nos permitirán realizar un seguimiento más real del proyecto cada poco tiempo, permitiendo reajustar aspectos como:

- Planificación del proyecto: Permite modificar la estimación de la duración de las tareas o el esfuerzo total que se puede realizar durante cada sprint.
- Detalles de la aplicación: Permite recodificar componentes ya acabados del sistema para adecuarlos a las nuevas necesidades de la aplicación, así como añadir funcionalidades nuevas que no se contemplaran en un principio pero son interesantes para el cliente.

Se ha optado por emplear esta metodología debido al auge existente de las metodologías ágiles en el desarrollo empresarial y a todos los beneficios que estas prácticas proporcionan. Entre sus grandes beneficios, podríamos destacar la facilidad de reajuste y adaptación que proporcionan ciclos de desarrollo cortos, la obtención de un producto ejecutable y revisable al acabar cada ciclo o la flexibilidad que este estándar de trabajo proporciona.

3.2 Análisis

Para el análisis inicial del proyecto, se ha esbozado el esqueleto general de la aplicación, tanto conceptualmente como funcionalmente. Para esto, se ha comenzado con la creación de

un modelo conceptual de base de datos, donde se han modelado todas las entidades que en un principio iban a constituir la aplicación, así como sus atributos.

Posteriormente, se ha realizado un estudio de las funcionalidades principales que ofertaría la aplicación, organizándolas en historias de usuario. Estas funcionalidades, a su vez, se han relacionado con las entidades para crear un primer esqueleto de como se estructurarían dichas operaciones.

Como último paso de este análisis inicial, se realizaron prototipos básicos de pantallas, como primera aproximación a como iba a ser visualmente la interfaz del usuario. Estos modelos se emplearon durante la codificación, cómo primera aproximación visual a la interfaz cliente.

Al tratarse de una metodología de trabajo basada en ciclos completos, antes de empezar cada sprint, también se realiza un análisis rápido, de cómo van a ser los requisitos seleccionados para dicho ciclo. Estos análisis son necesarios debido a la flexibilidad y adaptación que presenta el sistema al seguir un desarrollo ágil, ya que surgen diferentes casuísticas, quizás no contempladas en el análisis inicial.

3.3 Diseño

Tras un análisis inicial, utilizamos todos los componentes desarrollados durante esa fase para comenzar el diseño de la aplicación. Durante esta fase, transformamos las historias de usuario recogidas en requisitos funcionales más concretos que llevaremos a cabo durante la fase de implementación.

A través de estos requisitos determinamos la arquitectura más óptima para el sistema, así como las tecnologías que emplearemos para codificarlo. Esta fase del desarrollo también permite completar o actualizar las estructuras de datos, interfaces u otros componentes creados durante la fase de análisis.

Además, antes de comenzar un sprint existirá una fase de diseño rápida homóloga a la fase de análisis.

3.4 Implementación y pruebas

Tras las fases de análisis y diseño previas a cada ciclo, comenzaría la implementación correspondiente a los requisitos seleccionados. Estos requisitos serían codificados siguiendo una jerarquía de mayor a menor importancia, para asegurarnos de tener listas, al finalizar cada ciclo, el mayor número de funcionalidades críticas posibles. Esta selección se realizará a través de la priorización de tareas, que previamente habríamos creado, de todas las tareas existentes en el sprint backlog.

Esta jerarquía es importante, ya que a diferencia de metodologías más clásicas, esta me-

todología aboga por nunca aumentar las horas de trabajo dedicadas durante un sprint, si no por realizar una planificación cada vez mejor y más ajustada a la realidad. Por lo tanto, tareas programadas para un sprint pueden acabar siendo terminadas en el siguiente y estas tareas no terminadas deberían ser siempre las menos prioritarias.

Para cada funcionalidad nueva o modificada durante el ciclo, se diseñaron y codificaron pruebas para comprobar su correcto funcionamiento. Estas pruebas serían siempre de integración, cuyo objetivo sería comprobar que todos los componentes que intervienen en cada operación funcionen correctamente y devuelvan el resultado esperado.

3.5 Herramientas utilizadas

Durante la fase de análisis, se ha utilizado el editor de diagramas on-line **Draw.io** para la creación tanto de los diagramas UML como de todos los prototipos de pantallas. Para el diseño de las tarjetas correspondientes a las historias de usuario, se ha utilizado la aplicación **Trello** [23]. Para el diseño e implementación de las pruebas de integración de los servicios, se ha empleado la aplicación **Soap-UI**. Además, para la gestión de los sprints, tareas a realizar y la progresión de trabajo, nos hemos ayudado del gestor de proyectos **GitLab**, desplegado y mantenido por el grupo de investigación de los directores de este trabajo, el Laboratorio de Bases de Datos.

Capítulo 4

Planificación y seguimiento

4.1 Tareas y recursos

Como hemos comentado anteriormente, el desarrollo del proyecto se ha estructurado en sprints de desarrollo de una duración de entre 2-3 semanas, donde las tareas a realizar en cada uno se seleccionaban basándose en una escala de prioridad. A continuación, expondremos las tareas que han compuesto el proyecto. Para obtener estas tareas, durante la fase de análisis hemos deconstruido cada historia de usuario en casos de uso. Esos casos de uso se han agrupado lógicamente para crear cada una de las tareas de implementación que expondremos a continuación. Profundizaremos en el proceso anterior en la Sección 5.2.

HU-1: Login y gestión de usuarios

- 1.1 Creación del perfil de usuario.
- 1.2 Modelado de la entidad usuario y sus operaciones en el servidor.
- 1.3 Creación de páginas de Login/Registro.

HU-2: Gestión de méritos

- 2.1 Modelado de la entidad Mérito y sus operaciones en el servidor.
- 2.2 Modelado de las operaciones en cliente y la visualización de méritos.
- 2.3 Modelado de las entidades Tipos/Campos de méritos y sus operaciones en el servidor.
- 2.4 Refactorización de méritos para adecuarlos a la parametrización JSON.

HU-3 Gestión de la validación y la seguridad

- 3.1 Creación de validación de formularios y control de errores.
- 3.2 Codificación de seguridad básica HTML en cabeceras.

- 3.3 Separación del código de cliente y servidor.
- 3.4 Codificación de seguridad basada en tokens JWT.

HU-4 Operaciones de administrador

- 4.1 Creación de vista de administrador.
- 4.2 Modelado de la entidad Funciones en servidor y sus operaciones.
- 4.3 Codificación de editor de funciones JavaScript con comprobación de resultado instantáneo.

HU-5 Exportación de Méritos

- 5.1 Modelado de las entidades plantillas y secciones en servidor, así como sus operaciones.
- 5.2 Creación de funcionalidades para plantillas y secciones en cliente.
- 5.3 Codificación de exportación de currículums.
- 5.4 Modelado de plantillas Jasper para la creación del currículum de solicitud de plazas de la UDC.
- 5.5 Modelado de la entidad Documentos y sus operaciones.
- 5.6 Creación y aplicación de manuales de creación de plantillas.

4.2 Planificación inicial y coste

Para realizar la planificación de todas las tareas anteriormente descritas, lo primero es la construcción del llamado *product backlog*, al que nos hemos referido anteriormente en la Sección 3.1. Tras recopilar las tareas, deberemos realizar la estimación de cada una a través de la métrica de puntos de historia.

Esta métrica de puntos de historia es una manera de cuantificar el esfuerzo y la complejidad de las historias de usuario y de las tareas que la componen. Así, cuanto más compleja sea una historia o más esfuerzo requiera, más puntos de historia tendrá. La asignación de puntos de historia se realiza por medio de la escala de Fibonacci modificada (1, 2, 3, 5, 8, 13...), siguiendo la estrategia que hemos descrito anteriormente. La elección de utilizar este tipo de estimación es debido a la mayor facilidad de estimar basándose en la comparación de los tamaños de las tareas que en la mera carga de horas individual de cada una. Además, con esta estrategia evitamos el llamado factor optimista, que surge en estimaciones basadas en horas, ya que tendemos a minimizar la complejidad y tamaño de las tareas.

Tras esta estimación, introduciremos las tareas más prioritarias dentro del *sprint backlog* o lista de tareas del sprint, en función de los puntos de historia que hayamos acordado realizar.

El esfuerzo que será realizado durante cada sprint puede variar, ayudándose de las reuniones de retrospectiva realizadas al finalizar cada sprint. Además, el esfuerzo de cada sprint ha ido variando conforme a la disponibilidad de tiempo para dedicar al desarrollo.

4.3 Seguimiento

4.3.1 Organización de las tareas

A continuación, desglosaremos cómo se han repartido todas las tareas descritas anteriormente en los diferentes *sprints*, así como las fechas entre las que han tenido lugar:

Inicio del proyecto: El proyecto comienza el 4 de Abril de 2018.

Sprint 1 (4 Abril - 17 Abril): En este sprint, se ha realizado el proceso de análisis y diseño inicial. Para ello, se han creado diagramas UML tanto de las funcionalidades deseadas como del modelo conceptual. Además, se han creado las historias de usuario y los prototipos iniciales de pantalla.

Sprint 2 (18 Abril - 5 Mayo) : En este sprint, se ha creado el arquetipo del proyecto y se ha gestionado la persistencia de las primeras entidades. Además, se han modelado en el servidor las entidades de usuario y mérito, así como las funcionalidades de creación, borrado y actualización de las mismas.

Sprint 3 (6 Mayo - 25 Mayo): Este sprint se emplea en el aprendizaje del frameWork React y se crea la primera aproximación a la interfaz de usuario del cliente. La aplicación cliente muestra únicamente una serie de méritos propiedad del usuario.

Sprint 4 (26 Mayo - 10 Junio): Este sprint se emplea para desarrollar en el cliente las funcionalidades de gestión básicas de los méritos, así como el control de registro/ingreso de usuarios.

Sprint 5 (11 Junio - 24 Junio): Durante el sprint, se diseña un perfil de usuario editable, así como una barra de navegación para exponer las funcionalidades existentes.

Sprint 6 (25 Junio - 7 Julio): El ciclo se emplea en la creación de nuevas clases, que den soporte a la parametrización de los méritos. Se modelan entidades para definir el Tipo de Mérito, así como para cada uno de sus campos.

Interrupción 1 (8 Julio - 22 Septiembre): Por falta de disponibilidad de tiempo, el desarrollo no avanza durante estos dos meses.

Sprint 7 (23 Septiembre - 7 Octubre): El desarrollo vuelve a la normalidad y durante este sprint, se da soporte a las operaciones básicas para adecuarnos al nuevo paradigma parametrizado de los méritos, creado durante el último ciclo. Esto incluye operaciones básicas de gestión para tipos y campos de los méritos, así como nuevos creadores de méritos.

Sprint 8 (7 Octubre - 22 Octubre): Durante este sprint, se crean nuevas clases para gestionar la persistencias de las plantillas y secciones, que conformarán los currículums. Además de esto, se crea una primera plantilla simple con la librería JasperReports, que se rellena con la información proporcionada por el usuario, además del código en el servidor que permitirá la exportación de los currículums.

Interrupción 2 (28 Octubre - 15 Enero): Por falta de disponibilidad de tiempo, el desarrollo no avanza nuevamente durante estos dos meses.

Sprint 9 (16 Enero - 6 Febrero): Este sprint se aprovecha para refactorizar visualmente la aplicación, y añadir las funcionalidades de gestión, tanto para las plantillas, como para las secciones Jasper. Además se aprovecha para gestionar la validación de formularios y el control de errores.

Sprint 10 (7 Febrero - 18 Febrero): El sprint se destina íntegramente a la creación de una interfaz para el administrador de la aplicación. Así, todas las funcionalidades son divididas entre los dos tipos de interfaz y se crean algunas funcionalidades propias del administrador. Además, se realiza la separación entre el código del cliente y el servidor.

Sprint 11 (19 Febrero - 27 Febrero): Durante este sprint, se gestiona la seguridad a través del estándar JSON Web Token y se modela la entidad Función, correspondiente a las funciones JavaScript personalizadas, que puede crear el administrador. Además, se añade un editor que permite no solo crear dichas funciones, sino también ver el resultado de su aplicación.

Sprint 12 (28 Febrero - 13 Marzo): El sprint se destina a la creación de las plantillas Jasper, necesarias para la exportación del currículum de solicitud de plaza para la UDC y al modelado de la entidad Documento, que permitirá adjuntar archivos a cada mérito. Además, se añaden manuales de ayuda al usuario on-line, para la creación de currículums.

Como comentábamos en la sección anterior, la carga de cada uno de los sprints se ha organizado en función del tiempo disponible para la realización del mismo. Es por esto por lo que observamos que al inicio del proyecto realizamos unos *sprints* con muy poca carga de puntos historias de usuario y, sin embargo, los últimos *sprints* presentan una

carga de esfuerzo muy superior, por tener una disponibilidad completa al desarrollo del sistema. En la figura 4.1 presentamos un desglose de todas las tareas del proyecto, ordenadas en historias de usuario, con su comienzo y fin del desarrollo, así como la estimación de puntos de historia que se les asignó.

	Nombre de la tarea	Puntos de historia	Comienzo	Fin
	Proyecto Completo	110	4.04.18	13.03.19
1	HU-1: Login y gestión de usuarios	11	18.04.18	24.06.18
1.1	Creación perfil usuario.	3	11.06.18	24.06.18
1.2	Modelado entidad usuario y sus operaciones en el servidor.	5	18.04.18	24.04.18
1.3	Creación páginas de Login/Registro.	3	30.05.18	10.05.18
2	HU-2: Gestión de méritos	23	25.04.18	7.10.18
2.1	Modelado de la entidad Mérito y sus operaciones en el servidor.	8	25.04.18	5.05.18
2.2	Modelado de las operaciones en cliente y la visualización de méritos.	5	5.05.18	30.05.18
2.3	Modelado de las entidades Tipos/Campos de méritos y sus operaciones en el servidor.	5	25.06.18	7.07.18
2.4	Refactorización de méritos para adecuarlos a la parametrización JSON.	5	23.09.18	7.10.18
3	HU-3 Gestión de la validación y la seguridad	16	13.2.19	21.2.19
3.1	Creación de validación de formularios y control de errores.	2	2.1.19	6.01.19
3.2	Codificación de seguridad básica HTML en cabeceras.	1	17.2.19	18.2.19
3.3	Separación del código de cliente y servidor.	5	13.2.19	17.2.19
3.4	Codificación de seguridad basada en tokens JWT.	8	19.2.19	21.2.19
4	HU-4 Operaciones de administrador	19	7.2.19	27.2.19
4.1	Creación de vista de administrador.	5	7.2.19	12.2.19
4.2	Modelado de la entidad Funciones en servidor y sus operaciones.	1	21.2.19	22.2.19
4.3	Codificación de editor de funciones JavaScript con comprobación de resultado instantáneo.	13	22.2.19	27.2.19
5	HU-5 Exportación de Méritos	48	7.10.18	13.3.19
5.1	Modelado de las entidades plantillas y secciones en servidor así como sus operaciones.	5	7.10.18	17.10.18
5.2	Creación de funcionalidades para plantillas y secciones en cliente.	5	16.1.19	2.2.19
5.3	Codificación de exportación de currículums.	8	18.10.18	22.10.18
5.4	Modelado de plantillas Jasper para la creación del currículum de solicitud de plazas de la UDC.	21	28.2.19	8.3.19
5.5	Modelado de la entidad Documentos y sus operaciones.	5	8.3.19	11.3.19
5.6	Creación y aplicación de manuales de creación de plantillas.	1	12.3.19	13.3.19

Figura 4.1: Cuadro de la distribución de tareas en el tiempo y su estimación inicial

4.3.2 Desviaciones en tiempo y coste

Como hemos podido ver en el desglose anterior, las grandes desviaciones en tiempo del sistema han venido dadas por interrupciones del desarrollo, donde no se ha podido avanzar con la codificación de la aplicación.

Al haber realizado una organización más basada en el tiempo disponible que en un coste de esfuerzo fijo, las tareas introducidas en cada sprint se han podido realizar sin desviaciones reseñables, únicamente teniendo que realizar pequeños ajustes en las mismas durante el sprint posterior.

La duración final del proyecto ha sido de unos 11 meses y 1 semana. Las interrupciones del desarrollo han supuesto aproximadamente 5 meses y una semana, tiempo que podríamos contabilizar como la desviación temporal del objetivo marcado, siendo este el mes de Octubre.

En cuanto a las desviaciones en coste de las tareas, no se han encontrado diferencias reseñables que hayan podido provocar que el esfuerzo asignado a un sprint fuese inabarcable para el tiempo fijado. Por lo tanto, podríamos determinar que la estimación, a pesar de ser subjetiva, ha sido bastante acertada y ha permitido una organización eficiente del trabajo.

Capítulo 5

Análisis

5.1 Introducción

En este capítulo, vamos a profundizar en el proceso de recogida de información en forma de historias de usuario hasta su transformación y organización en las tareas anteriormente descritas en el capítulo 4. Además, veremos como se han construido las especificaciones del sistema, antes de entrar en una fase más detallada como veremos en el Capítulo 6.

5.2 Historias de usuario

La primera fase del análisis es la creación de las historias de usuario. Las historias de usuario son descripciones de las funcionalidades que aportan valor al usuario. Estas historias de usuario se suelen almacenar como tarjetas y presentan unos criterios de aceptación que deberán ser validados posteriormente. A continuación, mostramos cada una de las historias de usuario que se han creado para el desarrollo.

HU-1: Login y gestión de usuarios: Como usuario quiero:

- Poder crear un usuario y acceder a través de él a la aplicación.

Criterios de aceptación:

- Poder crear un perfil personal e individual a través de mis datos personales.
- Acceder a la aplicación y a sus funcionalidades a través de un login y una contraseña establecida por el propio usuario.
- Todos los datos introducidos podrán ser leídos, modificados o eliminados en cualquier momento.

HU-2: Gestión de méritos: Como usuario quiero:

- Poder leer, eliminar, editar o incluir cualquier mérito en la aplicación.
- Poder adjuntar documentos PDF a cualquier mérito.

Criterios de aceptación:

- Poder crear cualquier mérito y asociarlo a mi perfil de usuario.
- Poder añadir documentos PDF a la aplicación y asociarlos a los méritos ya creados.
- Todos los datos personales almacenados podrán ser leídos, actualizados o eliminados.

HU-3 Gestión de la validación y la seguridad: Como usuario quiero:

- Poder conocer si cualquiera de mis operaciones ha tenido éxito o si por lo contrario, ha surgido algún error.
- Que toda la información que proporcione a la aplicación y las operaciones que realice no puedan ser consultadas por otros usuarios.

Criterios de aceptación:

- Todas las operaciones realizadas por el usuario deben incluir algún tipo de alerta o señal que indique al usuario, una vez que la operación ha acabado, si lo ha hecho satisfactoriamente o, en caso de haber ocurrido algún error, lo informe.
- Todos los datos personales almacenados, así como las operaciones realizadas deberán estar securizadas y/o codificadas.

HU-4 Operaciones de administrador: Como administrador quiero:

- Poder crear, leer, modificar y eliminar los tipos de méritos y sus campos.
- Poder crear, leer, modificar y eliminar las plantillas de currículums y sus secciones.
- Poder crear funciones JavaScript que transformen a los méritos ya existentes o creen nuevos tipos de méritos. Además, quiero poder consultar, editar y eliminar estas funciones.

Criterios de aceptación:

- Poder crear, eliminar, leer y modificar los tipos de méritos y sus campos.
- Poder crear, eliminar, leer y modificar las plantillas y sus secciones.
- Poder comprobar que las funciones JavaSript son correctas y cuál será el resultado de su aplicación.
- Poder crear, eliminar, leer y modificar las funciones JavaScript.

HU-5 Exportación de Méritos: Como usuario quiero:

- Poder exportar los méritos incluidos en mi perfil, según diferentes plantillas, a documentos de texto descargables.

Criterios de aceptación:

- El usuario podrá seleccionar el formato en el que desea exportar su currículum, dentro de todas las plantillas proporcionadas por la aplicación.
- El usuario dispondrá de manuales de ayuda consultables para ayudarle en la creación de su currículum.

5.3 Requisitos del sistema

Tras crear las tarjetas con las historias de usuario, el siguiente paso es su transformación en casos de uso que nos permitan desgranar cada tarea para facilitar su análisis, diseño y posterior planificación. En la Tabla 5.3 podemos observar como a partir de cada historia de usuario, hemos obtenido numerosos casos de uso. Además, en la Sección 5.3.2 veremos los requisitos no funcionales que también han surgido del análisis de estas historias de usuario.

Transformación de historias de usuario en funcionalidades	
HU1: Login y Gestión de usuarios	Crear/Editar/Eliminar/Consultar usuario Login/Registro usuario
HU-2: Gestión de méritos	Crear/Editar/Eliminar/Consultar méritos Crear/Editar/Eliminar/Consultar documentos
HU-3 Gestión de la validación y la seguridad	
HU-4 Operaciones de administrador	Crear/Editar/Eliminar/Consultar tipos de mérito Crear/Editar/Eliminar/Consultar campos de mérito Crear/Editar/Eliminar/Consultar plantillas Crear/Editar/Eliminar/Consultar secciones Crear/Editar/Eliminar/Consultar funciones
HU-5 Exportación de Méritos	Consultar manuales Creación y exportado de currículums

Cuadro 5.1: Relación entre historias de usuario y casos de uso

5.3.1 Requisitos funcionales

Los requisitos funcionales están divididos en dos grandes grupos, correspondiéndose con las funcionalidades propias de los usuarios comunes y el administrador.

Los requisitos funcionales del usuario, a su vez, se podrían dividir en las funcionalidades que se pueden realizar sin estar registrado en la aplicación y aquellas que no. Al tratarse de un sistema que opera sobre datos relacionados con el usuario y su exportación, el usuario no registrado tiene muy limitadas las funcionalidades disponibles:

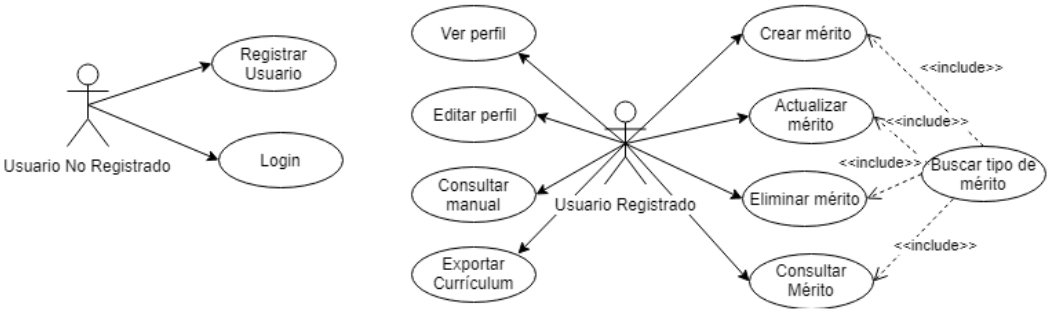

Figura 5.1: Casos de uso de los requisitos funcionales del usuario

Registro de usuario: El usuario no registrado podrá crear un perfil de usuario que le permitirá ingresar en la aplicación y acceder a sus funcionalidades.

Login: El usuario no registrado, a través de un nombre de usuario y una contraseña, podrá acceder a las funcionalidades de la aplicación.

Una vez el usuario ha ingresado en la aplicación, se le presentan las siguientes funcionalidades:

Creación/Consulta/Edición/Borrado de méritos: El usuario podrá incluir un mérito de cualquier tipo disponible, así como actualizarlo, borrarlo o simplemente consultarlo. Para esto, primero deberá seleccionar el tipo de mérito con el que desea interaccionar.

Consulta/Edición del perfil de usuario: El usuario podrá acceder a su información personal en cualquier momento, así como modificarla.

Consultar Manual: El usuario podrá consultar el manual de ayuda a la creación de cada formato de currículum, previamente seleccionando el formato a consultar.

Exportar currículum: El usuario podrá exportar el currículum del formato que desee, dentro de los ofertados por la aplicación, únicamente seleccionándolo. El currículum se rellenará con los méritos añadidos previamente por el usuario.

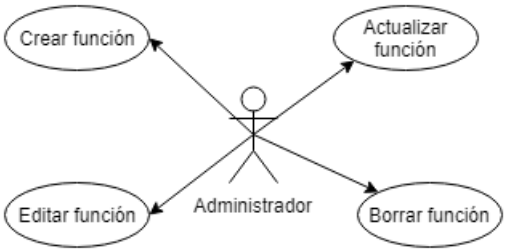

Figura 5.2: Casos de uso de los requisitos funcionales del administrador (1)

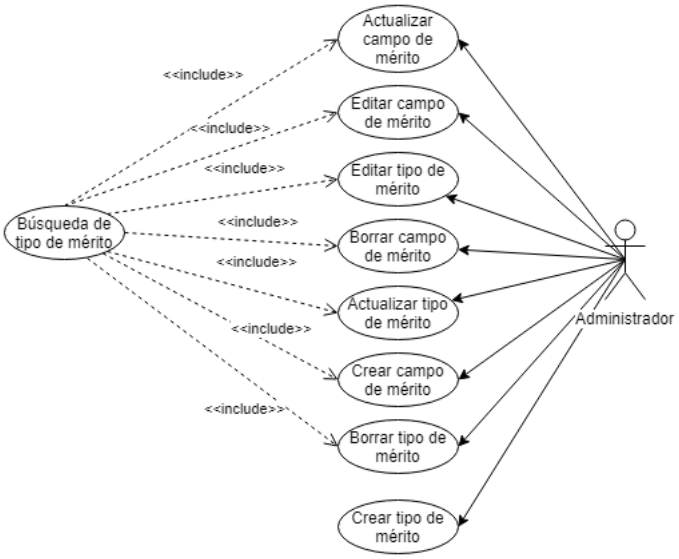

Figura 5.3: Casos de uso de los requisitos funcionales del administrador (2)

La aplicación, a su vez, también presenta un vista de administrador con funcionalidades exclusivas para este. Estos requisitos funcionales son:

Creación/Consulta/Edición/Borrado de tipo de mérito: El administrador podrá crear nuevos tipos de méritos que pasarán a estar disponibles para los usuarios. Además, podrá editarlos, consultarlos o borrarlos previamente seleccionando el tipo de mérito sobre el que operar.

Creación/Consulta/Edición/Borrado de campo de mérito: El administrador podrá añadir nuevos campos a un tipo de mérito ya creado. Estos nuevos campos pasarán a estar disponibles a los usuarios para cumplimentar al crear un mérito de ese tipo. Además, podrá editarlos, consultarlos o borrarlos, previamente seleccionando el tipo de mérito sobre el que operar.

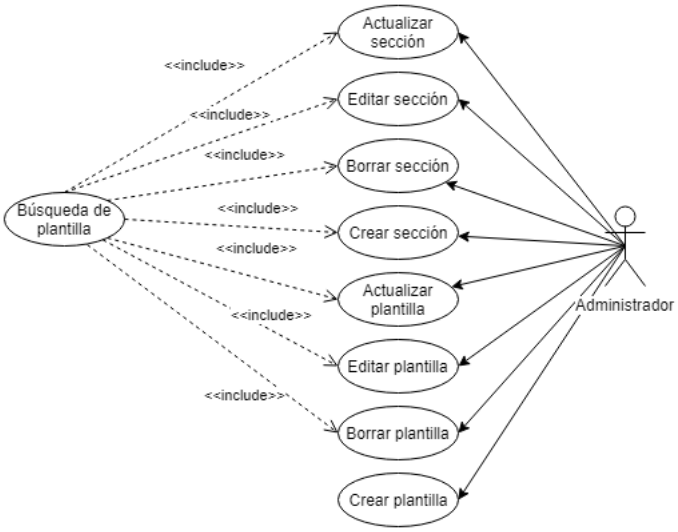

Figura 5.4: Casos de uso de los requisitos funcionales del administrador (3)

Creación/Consulta/Edición/Borrado de plantillas: El administrador podrá crear nuevas plantillas de currículums, que pasarán a estar disponibles para los usuarios como nuevo formato de exportación. Además podrá editarlas, consultarlas o borrarlas, previamente seleccionando la plantilla sobre la que operar.

Creación/Consulta/Edición/Borrado de secciones: El administrador podrá añadir nuevas secciones a una plantilla ya creada. Estas nuevas secciones formarán parte del exportado de currículums de ese formato. Estas secciones podrán ser editadas, consultadas o borradas, previamente seleccionando la plantilla deseada.

Creación/Consulta/Edición/Borrado de funciones: El administrador podrá crear funciones personalizadas JavaScript para adaptar tipos de méritos existentes, a otros formatos, o crear tipos nuevos. Para esto, existirá un editor que permitirá verificar la validez del código creado, así como un ejemplo del resultado de aplicar dicha función. Además, están funciones serán editables, borrables y consultables.

5.3.2 Requisitos no funcionales

Además de las funcionalidades esperadas por la aplicación, también se esperan otros requisitos en el comportamiento del sistema. Estos son:

Seguridad: La información de los usuarios debe ser confidencial y no debe permitirse el acceso a estos por parte de terceros. Además, todas las operaciones que realice un usuario sobre el sistema deben verificar la autenticidad y autorización que dicho usuario posee.

Escalabilidad: Uno de los pilares del sistema es la capacidad de ofertar un abanico de formatos que pueden incrementarse con el tiempo. Por eso, se espera del sistema que sea capaz de adaptarse a las nuevas inclusiones de formatos, sin necesidad de modificar el código.

Facilidad de uso: La aplicación debe presentar una interfaz simple y fácil para el usuario. Además, se proporcionarán manuales de ayuda on-line para facilitar la creación de currículums y ayuda en las funcionalidades más complejas.

5.4 Arquitectura del sistema

La aplicación se ha desarrollado siguiendo una arquitectura **cliente-servidor web**. Este patrón de arquitectura diferencia dos componentes principales: un proveedor de recursos (Servidor) y un acreedor de recursos (Cliente) que interactúan entre sí a través de peticiones sobre la red. Los componentes que forman la arquitectura son:

- **Cliente**: Interfaz de la aplicación accedida por el usuario final, que obtiene todos sus datos de peticiones realizadas al servidor.
- **Servidor**: Proveedor de recursos que recibe las peticiones de los clientes, se comunica con el SGBD para realizar las operaciones y devuelve los resultados a los clientes. Las funcionalidades proporcionadas por el servidor se ofertan dentro de una API.
- **SGBD**: Repositorio de información persistente, accedida únicamente por el servidor.

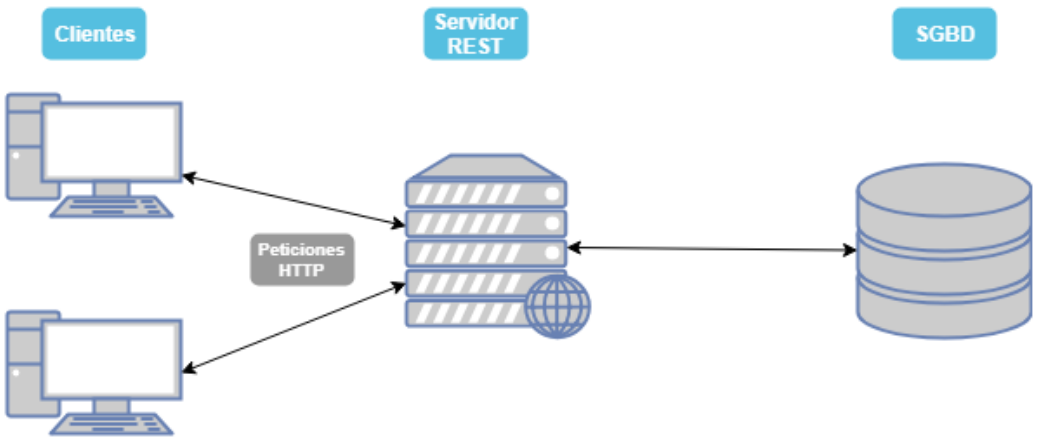

Figura 5.5: Diagrama de la arquitectura cliente-servidor web

5.5 Interfaz de usuario

La aplicación cliente se ha programado como una SPA, Single Page Application. Por este motivo, la aplicación consta de una página principal con unos componentes que se renderizan condicionalmente, apareciendo o no, en función de las necesidades del usuario.

El cliente presenta una barra de navegación que permite interaccionar con las diferentes funcionalidades y navegar entre ellas. Así mismo, existirán barras de búsqueda para seleccionar los tipos de datos que se quieren consultar. Una vez accedido al dato deseado, podremos consultar cualquier información y existirá un desplegable con todas las acciones que podremos realizar sobre dicho elemento.

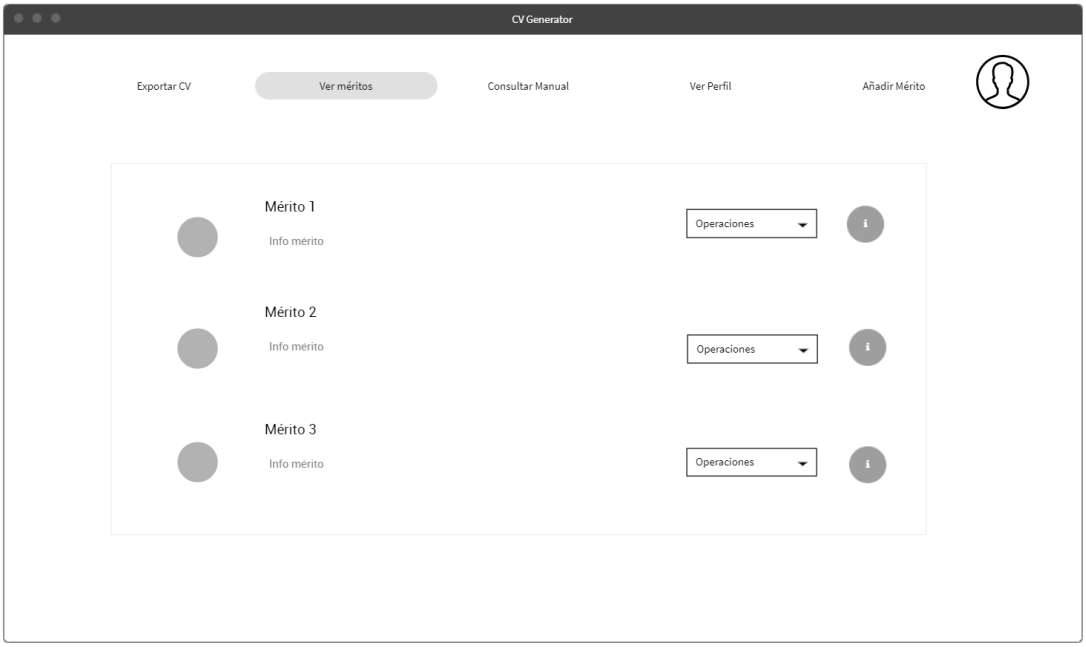

Figura 5.6: Prototipo de pantalla principal de la aplicación

En la Figura 5.6 podemos ver el prototipo diseñado para la vista principal del usuario. En ella vemos una lista de méritos del usuario que presentan un desplegable con todas las operaciones que podemos acometer contra ellos. Además presenta la barra de navegación descrita anteriormente para navegar entre todas las funcionalidades. Interactuar con las funcionalidades de edición, actualización y borrado de los datos, o la exportación de los mismos, provocará la aparición de diálogos para acometer dichas operaciones.

5.6 Modelo conceptual de datos

Como hemos comentado en la Sección 1.2, la finalidad del proyecto es la creación de un sistema completamente flexible, que permita la creación de cualquier tipo de estructura tanto para los méritos como para las plantillas de los currículums. Para lograr esto, hemos diseñado un modelo de datos que nos permita crear estructuras de datos personalizadas basadas en clases del propio sistema.

Dentro del modelo de datos podemos distinguir entre dos grandes grupos de clases: las clases principales, que contienen la información que será empleada en las funcionalidades del sistema y las clases secundarias, que serán las encargadas de determinar la estructura de algunas de estas clases principales. Podemos observar en la Figura 5.7 cuales son estas clases y las relaciones existentes entre ellas.

Como clases principales hemos modelado:

- **Account**: Clase que almacena los datos personales del usuario.
- **Merit**: Clase que almacena los méritos del usuario.
- **Document**: Clase que almacena los documentos anexos a un mérito.
- **Function**: Clase que almacena las funciones transformadoras de información
- **Report**: Clase que almacena la plantilla de creación de los currículums. Esta estará compuesta por un conjunto de secciones.

Como clases secundarias hemos modelado:

- **MeritType**: Clase que define un tipo de mérito. Estará compuesto por un conjunto de campos.
- **MeritField**: Clase que define un campo de un tipo de mérito.
- **SubReport**: Clase que almacena una sección de una plantilla.

Estas clases secundarias son las que permiten alcanzar nuestro objetivo de parametrizar la creación de currículums. Así, podremos determinar cualquier estructura de información que debe almacenar un mérito, simplemente creando un tipo de mérito y el conjunto de campos deseado. Así mismo, la plantilla que genera un currículum esta formada por cualquier conjunto de secciones, y estas, a su vez, están relacionadas con el tipo de mérito que define la estructura de datos que emplean. Esto en conjunto, permite que un currículum pueda presentar cualquier formato deseado e incluir, en dicho formato, cualquier información que precise.

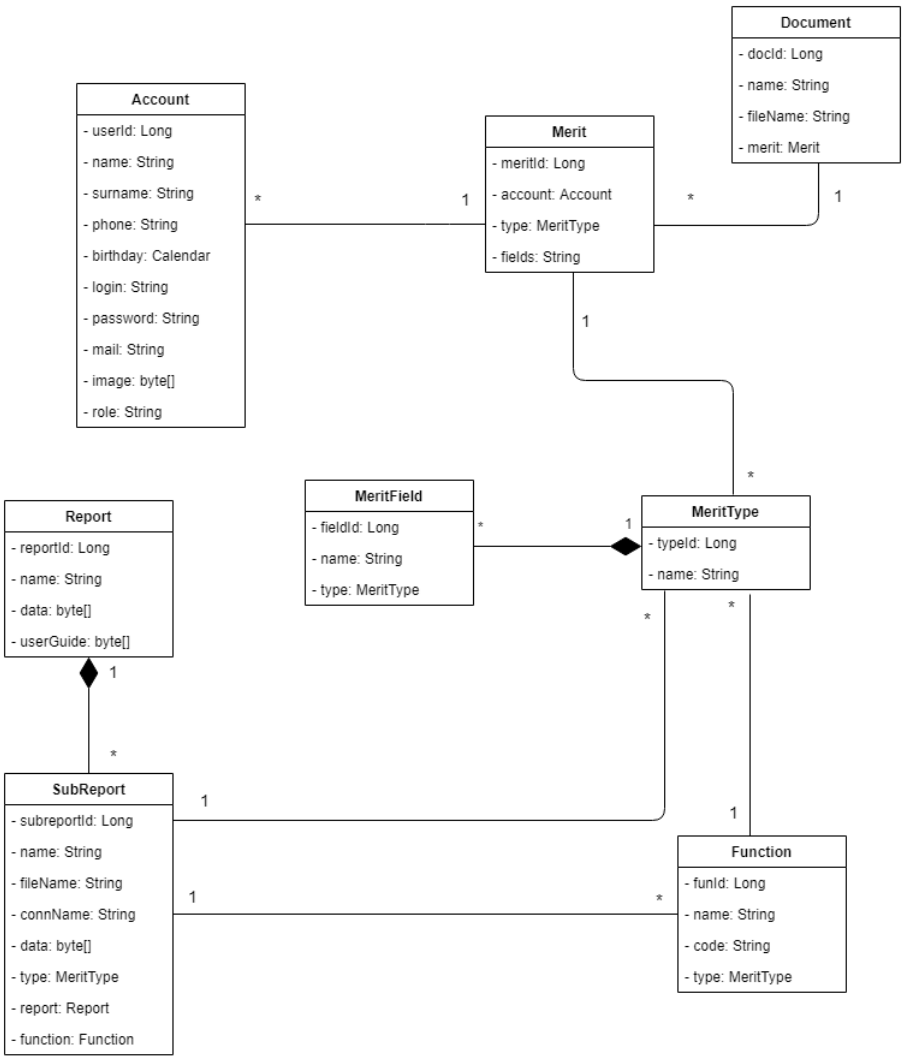

Figura 5.7: Diagrama de entidades persistentes

Capítulo 6

Diseño

6.1 Arquitectura tecnológica del sistema

En este capítulo analizaremos cómo se han creado los diferentes componentes del sistema, basándonos en el análisis que hemos realizado durante el capítulo anterior.

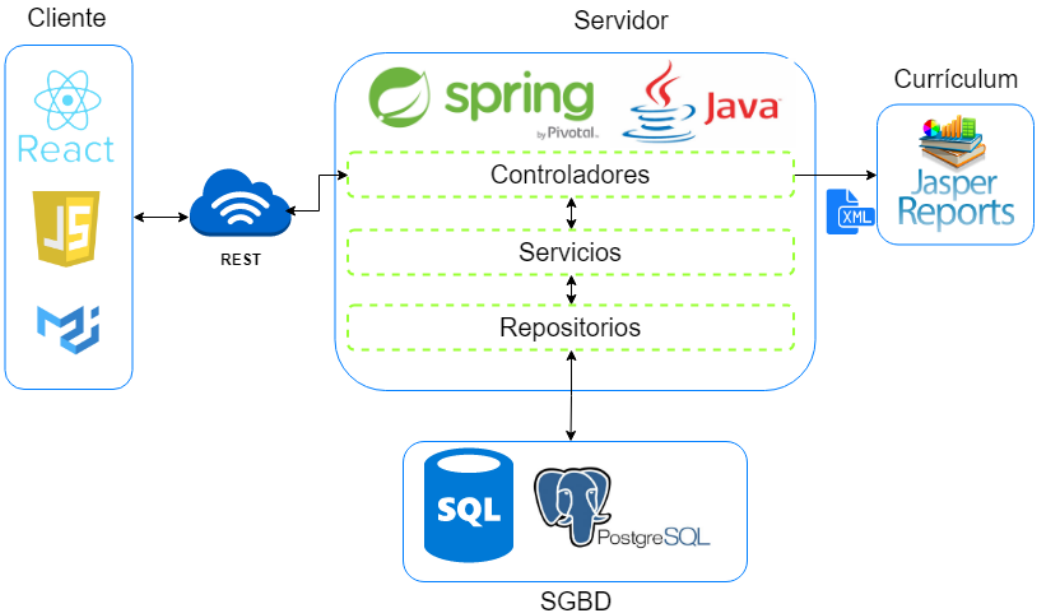

Figura 6.1: Arquitectura tecnológica del sistema

6.2 Cliente

Entendemos como cliente al conjunto de componentes diseñados para que el usuario pueda emplear las funcionalidades expuestas por el servidor, utilizando las tecnologías web. El cliente se ha modelado siguiendo el patrón SPA, es decir, está compuesto por una única página, donde los componentes se van creando o desapareciendo en función de las necesidades del usuario. Esto permite que la experiencia de usuario sea más fluida y rápida, ya que los componentes se generan dinámicamente y el usuario no debe navegar entre jerarquías de páginas, que pueden hacer que el uso de la aplicación llegue a ser tedioso. La codificación de los componentes se ha realizado a través del framework React, que permite la creación de componentes JavaScript inteligentes, que controlan su propio estado y comportamiento. Se ha optado por emplear React para la codificación del cliente, debido al auge de dicha tecnología en el entorno empresarial y a la atractiva gestión de componentes que se observó que realizaba durante la fase de análisis inicial del proyecto. Cada componente de la aplicación presenta una serie de aspectos similares:

- Comportamiento: Código encargado de obtener los datos del servidor o realizar las operaciones que haya pedido el usuario.
- Estado: Código encargado de gestionar el estado del componente, detectando cuándo se debe crear el componente, eliminar o actualizar su información.
- Renderización: Código encargado de gestionar cómo se mostrará dicho componente en cada momento, transformando el código React+JavaScript al HTML necesario para mostrarse en el navegador.
- Comunicación: El código React se basa en una jerarquía padre/hijos de componentes, donde cada componente puede hacer uso de la información que el componente que lo haya creado (padre) le haya proporcionado. Este componente padre gestionará no solo los datos que proporcionará a cada hijo (métodos expuestos y datos), sino cuándo estos componentes deben crearse o eliminarse.

6.2.1 Estructura de los ficheros

El paquete cliente alberga en su raíz, además de los ficheros necesarios de dependencias y estructura (package.json y webpack.config), dos elementos principales:

- html: En este paquete nos encontramos todos los elementos no pertenecientes a archivos JavaScript o React. Aquí encontramos el archivo html sobre el que se construye la aplicación cliente, además de sus estilos.

- js: Aquí se encuentran ordenados todos los archivos JavaScript que forman el cliente. Como no existe una jerarquía clara de estructuración global para los archivos React, hemos optado por realizar una jerarquía similar a Java, donde a cada familia de componentes principales los hemos incluido dentro de un paquete con su nombre. Dentro de cada uno de esos paquetes, nos encontramos los ficheros del componente que le da nombre al paquete, así como los de subcomponentes o diálogos que se generan a través del primero.

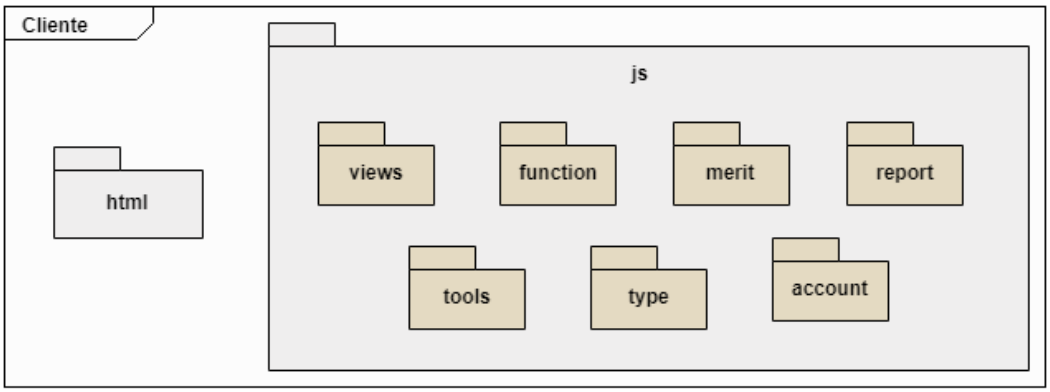

Figura 6.2: Diagrama de paquetes físicos del cliente

6.2.2 Estructura lógica

En la sección 6.2 hemos analizado la estructura general de cada uno de los componentes que conforman el cliente. Así mismo, hemos visto que estos componentes presentan una organización interna específica que marca las interacciones entre ellos. Cada componente dependerá de un padre, que será el encargado de pasarle la información necesaria que no sea la específica del componente, y controla cuándo el componente debe ser visible y cuándo no. En esta sección, observaremos cuáles son los componentes concretos que forman el cliente y la jerarquía establecida entre ellos.

Como podemos ver en la figura 6.3, el cliente se compone de un elemento principal llamado app, desde el que se construye toda la aplicación cliente. Existen tres vistas principales dentro de la aplicación, que se alternan dinámicamente en función de los privilegios del usuario o de si este se encuentra o no logueado en la aplicación. Estas vistas principales son:

Login: Vista del cliente que engloba a todos los componentes que permiten controlar el ingreso de los usuarios a la aplicación. Además, controla la vista que permite el registro de los usuarios, en caso de que todavía no posean una cuenta en la aplicación.

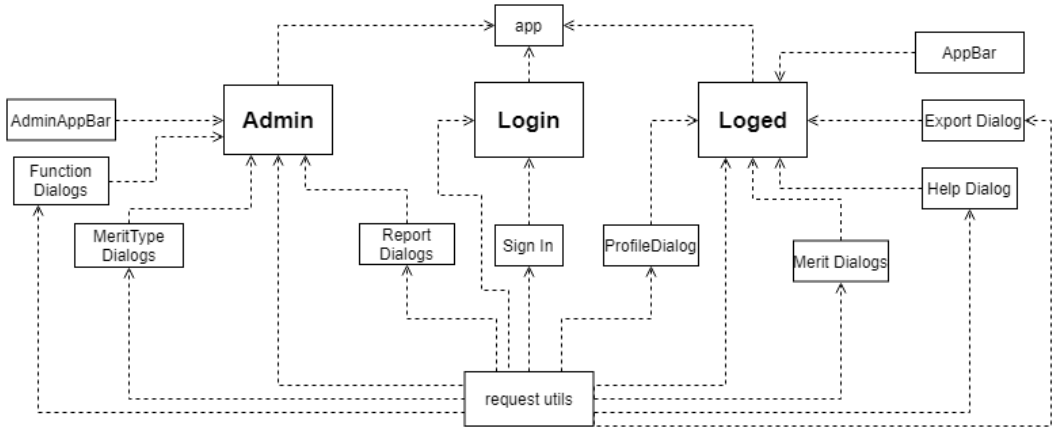

Figura 6.3: Diagrama simplificado de las clases que forman la aplicación cliente

Loged: Vista del cliente que engloba a todos los componentes a los que un usuario común tiene acceso. Estos componentes serán accesibles al usuario tras loguearse correctamente en la aplicación. Estos diálogos permitirán al usuario:

- La gestión de los méritos.
- La gestión del perfil de usuario.
- La consulta de manuales de ayuda.
- La exportación de currículums.

Admin: Vista del cliente que engloba a todos los componentes a los que únicamente un usuario con privilegios de administrador tiene acceso. Esta vista, a su vez, presentará tres vistas intercambiables, que se corresponden con los tres grandes grupos de funcionalidades:

- Gestión de tipos de mérito.
- Gestión de funciones.
- Gestión de plantillas.

Las vistas de Loged y Admin emplearán cada una una barra de navegación propia, que permitirá la navegación y el acceso a los diferentes diálogos de funcionalidades. Además, existe un componte global de utilidades que emplean la mayoría de componentes de la aplicación. Este componente es el encargado de gestionar las peticiones, construyéndolas y adecuándolas a los estándares de seguridad del servidor. Cualquier componente que necesite obtener o enviar información al servidor deberá emplear esta clase.

6.3 Servidor

Dentro del servidor, nos encontramos con un conjunto de componentes que permiten resolver las peticiones de los clientes y gestionar su persistencia a través de la comunicación con el SGBD.

Se ha optado por emplear la arquitectura REST para modelar la comunicación entre cliente y servidor. Este estándar pretende imitar la comunicación de los componentes dentro de Internet y adaptarlo para poder aplicarlo al desarrollo de sistemas. La arquitectura, como concepto principal, determina que toda la información que intercambien los clientes con el servidor será por medio de peticiones que seguirán el protocolo HTTP.

El servidor se ha codificado a través del lenguaje Java y el framework Spring-boot, que facilita la ejecución de patrones de diseño (fachada, inyección de dependencias...), la creación de componentes (repositorios, servicios y controladores), además de proporcionar facilidades de comunicación con el SGBD y la creación de la base de datos. Además, el servidor empleará la librería JasperReports para obtener el currículum final y exportarlo, para que el cliente pueda tanto visualizarlo como descargarlo para su almacenamiento.

El servidor está compuesto por tres tipos de clases que interactúan tanto entre sí, como con los componentes que se encuentran fuera del servidor. Estos componentes son:

Controlador: El controlador es el encargado de exponer las funcionalidades del servidor y de gestionar las peticiones de las aplicaciones cliente. Para cada petición, delegará en el servidor todas las operaciones que tenga que realizar y devolverá al finalizar una respuesta al cliente. Esta respuesta puede ser tanto un error, en caso de que haya ocurrido algo inusual, o la respuesta habitual acompañada de una señal de que todo ha terminado satisfactoriamente.

Servicio: Es la clase encargada de gestionar la lógica de las operaciones para cada una de las entidades persistentes. Recibirá los datos del controlador, realizará las operaciones necesarias y delegará en el repositorio la resolución de todas las llamadas necesarias al SGBD, tanto para obtener datos, como para actualizar, borrar o crear entidades.

Repositorio: Es la clase encargada de la comunicación con el SGBD. Su única función es la de transformar las llamadas del servicio a lenguaje SQL para poder comunicarse con la base de datos. Devolverá al servicio, al finalizar, toda la información que la base de datos haya proporcionado, transformándola inversamente a lenguaje Java.

6.3.1 Estructura de los ficheros

El paquete servidor se estructura de una manera parecida al cliente. Dentro de su raíz, nos encontramos el fichero principal del proyecto (pom.xml), así como una carpeta con documen-

tación. Dentro de los ficheros código propiamente dicho (src/main), nos encontraremos dos elementos principales:

resources: En este paquete nos encontramos todos los elementos no Java que participan en el proyecto. Aquí hemos incluido las plantillas Jasper.

java/cv: En este paquete, siguiendo una jerarquía Java usual, nos encontramos los paquetes principales del código Java. Cada uno de los paquetes recibe el nombre de la entidad que gestiona y contiene una clase de cada uno de los siguientes tipos:

- Clase principal: Clase de la entidad con sus atributos y métodos de acceso a datos.
- Repositorio: Clase que realiza las operaciones contra el SGBD.
- Servicio: Clase que conforma la lógica de la entidad y comunica a el controlador con el repositorio.
- Controlador: Clase que engloba las funcionalidades expuestas de la entidad, así como la comunicación para cada petición con el servicio.

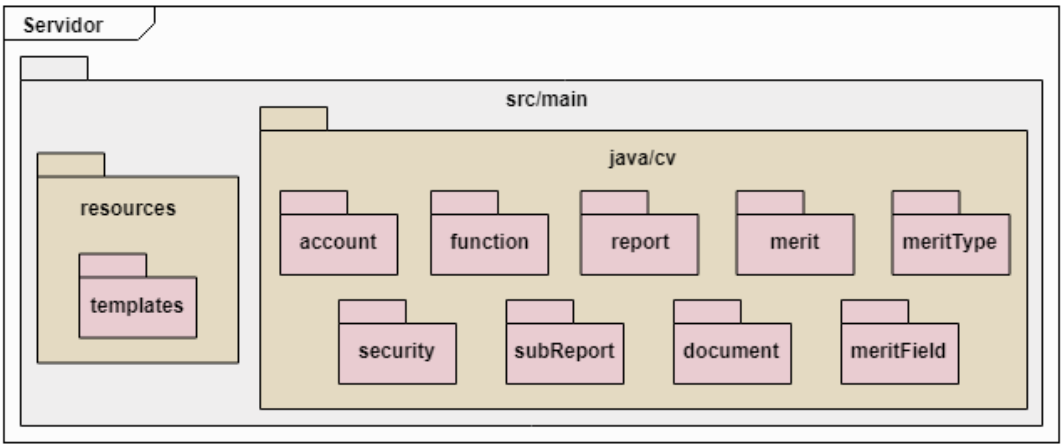

Figura 6.4: Diagrama de paquetes físicos del cliente

6.3.2 Relación entre casos de uso y operaciones del servidor

En el Capítulo 5 hemos visto como las historias de usuario se desfragmentaban en casos de uso. Ahora, en esta fase de diseño, estos casos de uso se han transformado en operaciones que serán accesibles a través de un controlador, como hemos comentado anteriormente. En las Tablas 6.1, 6.2, 6.3, 6.4, 6.5 podemos observar la relación de cada caso de uso en una operación expuesta por un controlador que será accesible para cualquier aplicación cliente. En la Sección 6.3.3 se expondrán en contexto cada una de estas operaciones.

Casos de uso Account Controller	
Crear usuario	create(Account): Account
Eliminar usuario	deleteUser(Long): void
Consultar usuario	getUser(Long): Account
Actualizar usuario	updateUser(Account, Long): Account
Login usuario	login(String, String): Account
Registro usuario	saveUsuario(String, String, String, String, String, MultipartFile)

Cuadro 6.1: Transformación de casos de uso a operaciones del servidor para la gestión del usuario

Casos de uso Merit Controller	
Crear mérito	addMerit(String): Merit
Eliminar mérito	deleteMerit(Long): void
Consultar méritos	getMeritsByType(Long): List<Merit> getAllMeritsByType(Long): List<Merit> getAllMeritsByTypeAndNumber(Long, int): List<Merit> getMeritsByTypePaged(Long, int, int): List<Merit>
Actualizar mérito	updateMerit(String, Long): Merit
Crear documento	addDocument(String, Long, MultipartFile): Document
Eliminar documento	deleteDocument(Long): void
Consultar documentos	getDocumentsByMerit(Long): List<Document>

Cuadro 6.2: Transformación de casos de uso a operaciones del servidor para la gestión de méritos y documentos

Casos de uso FunctionController	
Crear función	saveFunction(String, String, Long): Function
Eliminar función	deleteFunction(Long): void
Actualizar función	updateFunction(Long, string): Function
Consultar funciones	getAllFunctions(): List<FunctionDTO> getFunctionsByType(Long): List<Function>

Cuadro 6.3: Transformación de casos de uso a operaciones del servidor para la gestión de funciones

Casos de uso MeritTypeController	
Crear tipo de mérito	addType(String): MeritType
Eliminar tipo de mérito	deleteType(Long): void
Actualizar tipo de mérito	updateType(Long, String): MeritType
Consultar tipos de mérito	getTypes(): List<MeritType> getTypeById(Long): MeritType
Crear campo de mérito	addField(name, Long): MeritField
Eliminar campo de mérito	deleteField(Long): void
Actualizar campo de mérito	updateField(Long, String): MeritField
Consultar campos de mérito	getFields(): List<MeritField>

Cuadro 6.4: Transformación de casos de uso a operaciones del servidor para la gestión de tipos y campos de mérito

Casos de uso ReportController	
Crear plantilla	createReport(MultipartFile, String): Report
Eliminar plantilla	deleteReport(Long): void
Actualizar plantilla	updateReport(Long, String, MutipartFile): Report
Consultar plantillas Consultar manuales	getReports(): List<Report>
Crear sección	createSubReport(MultiPartFile, String, Long, Long, Long): SubReport
Eliminar sección	deleteSubReport(Long): void
Actualizar sección	updateSubReport(Long, String, MultipartFile): SubReport
Consultar secciones	getSubReportsByReport(Long): List<SubReportDto>
Exportar currículum	exportPDF(Long): ResponseEntity<byte[]>

Cuadro 6.5: Transformación de casos de uso a operaciones del servidor para la gestión de plantillas, secciones y exportado de currículums

6.3.3 Controladores, servicios y repositorios del servidor

En esta sección, podemos observar cuales son los métodos modelados en la aplicación para cada uno de los tipos de clases que intervienen en el comportamiento del servidor. Para cada una de las clases que conforman la aplicación (Figura 5.7), se ha creado un repositorio y servicio propio. En cuanto a los controladores se ha seguido una estrategia parecida, pero englobando las entidades débiles dentro de la entidad fuerte. Así las operaciones de documentos se incluyen dentro del controlador de méritos, las operaciones de secciones dentro del controlador de plantillas y, por último, las operaciones de los campos de mérito se incluyen en controlador de tipos de mérito.
Para poder entender el funcionamiento de todas estas estructuras vamos a poner como ejemplo las operaciones relacionadas con la cuenta del usuario.Desde una aplicación cliente cuando se quieren realizar operaciones con la cuenta del usuario,esta debe realizar una llamada a una de los métodos expuestos en el *Account Controller*. Cada una de estas operaciones delegará en una o varias de las operaciones del *Account Service*. En esta entidad, además, se realizará la lógica de negocio para cada operación. Por último cuando esta clase necesite persistir la información delegará en los métodos de *Account Repository*.

Por ejemplo, a la hora de registrar a un nuevo usuario, la aplicación cliente realiza una llamada al método **saveUser()** del *Account Controller* y le envía toda la información del nuevo usuario. Este método del controlador delegará en el método **save()** del *AccountService* que relizará la lógica necesaria y para su posterior persistencia en base de datos delegará a su vez en el método con el mismo nombre del *AccountRepository*.

Podemos ver el desglose de los repositorios y servicios en las Figuras 6.5, 6.6 y los controladores en la Figura 6.7.

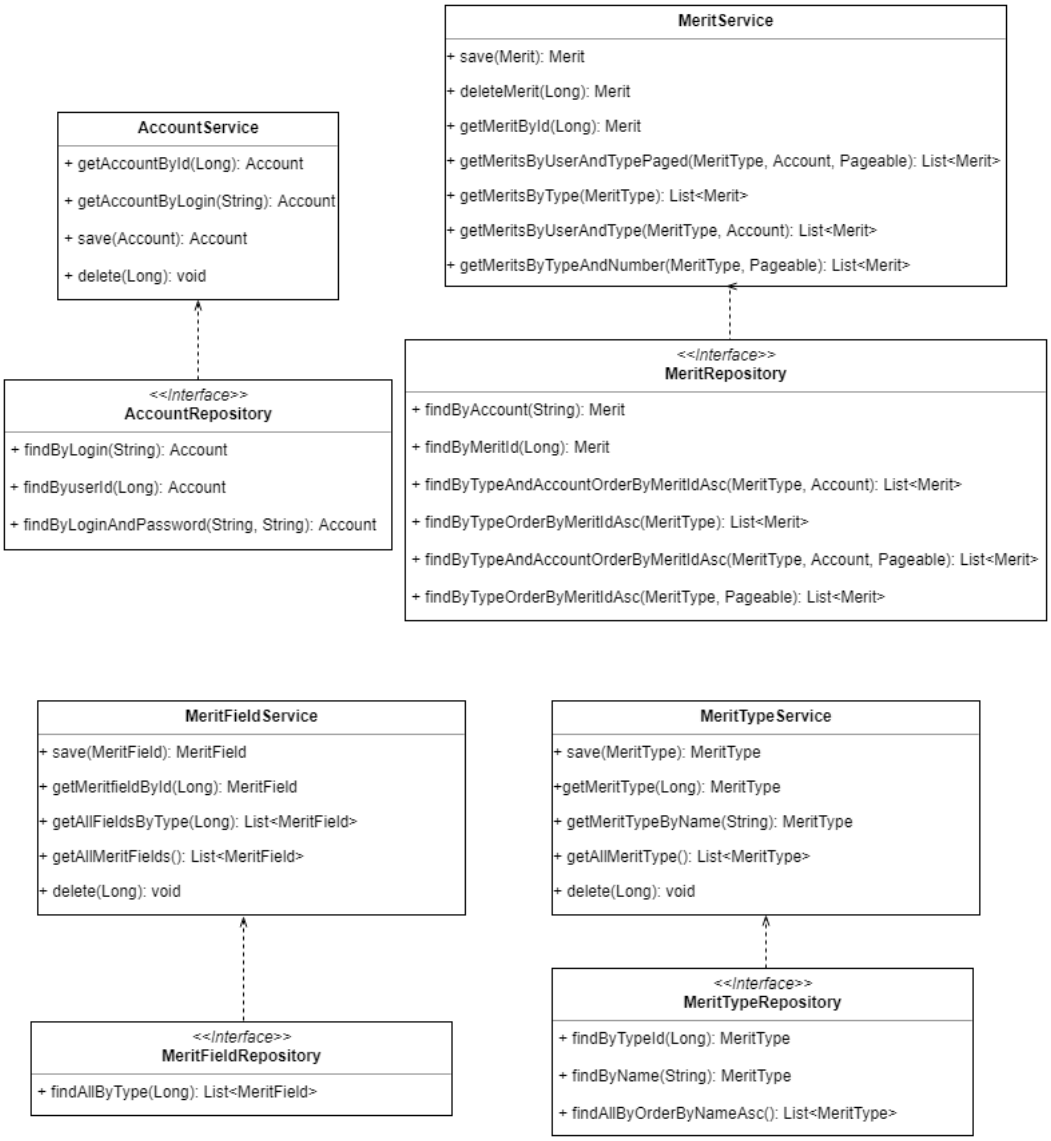

Figura 6.5: Diagrama de servicios y repositorios de las entidades persistentes (1)

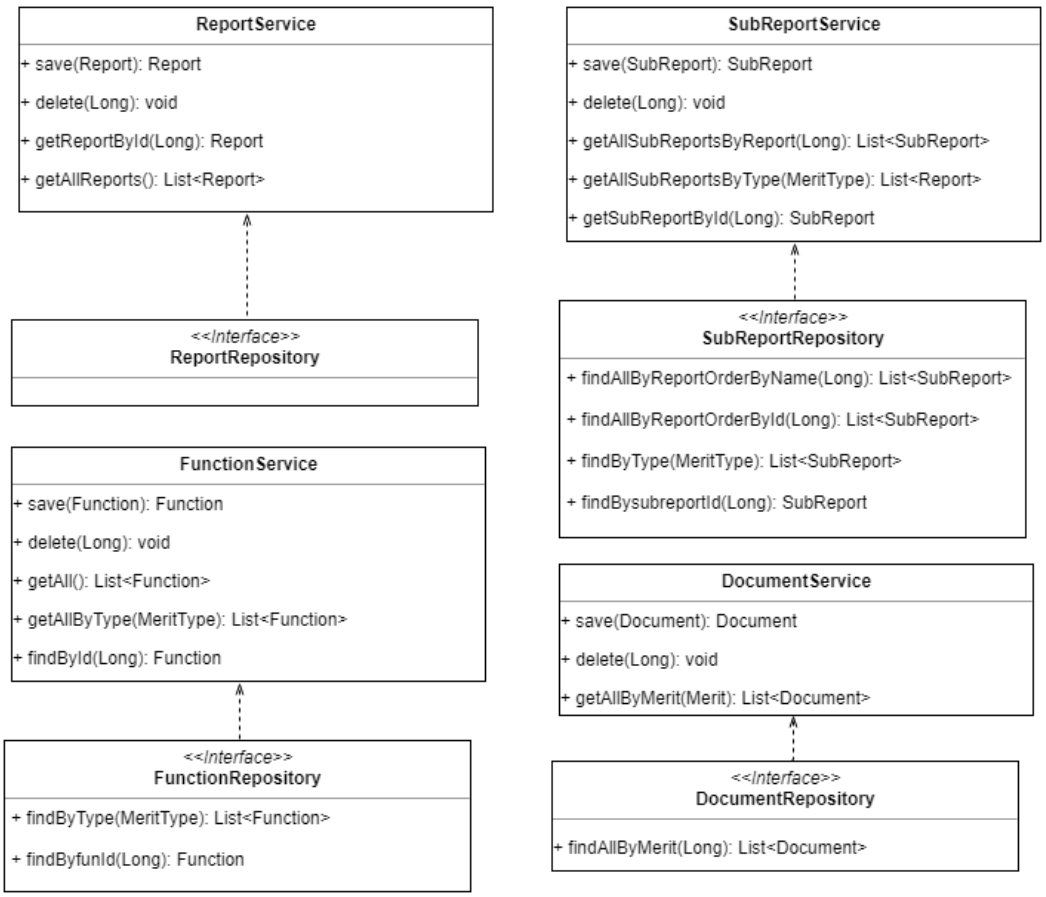

Figura 6.6: Diagrama de servicios y repositorios de las entidades persistentes (2)

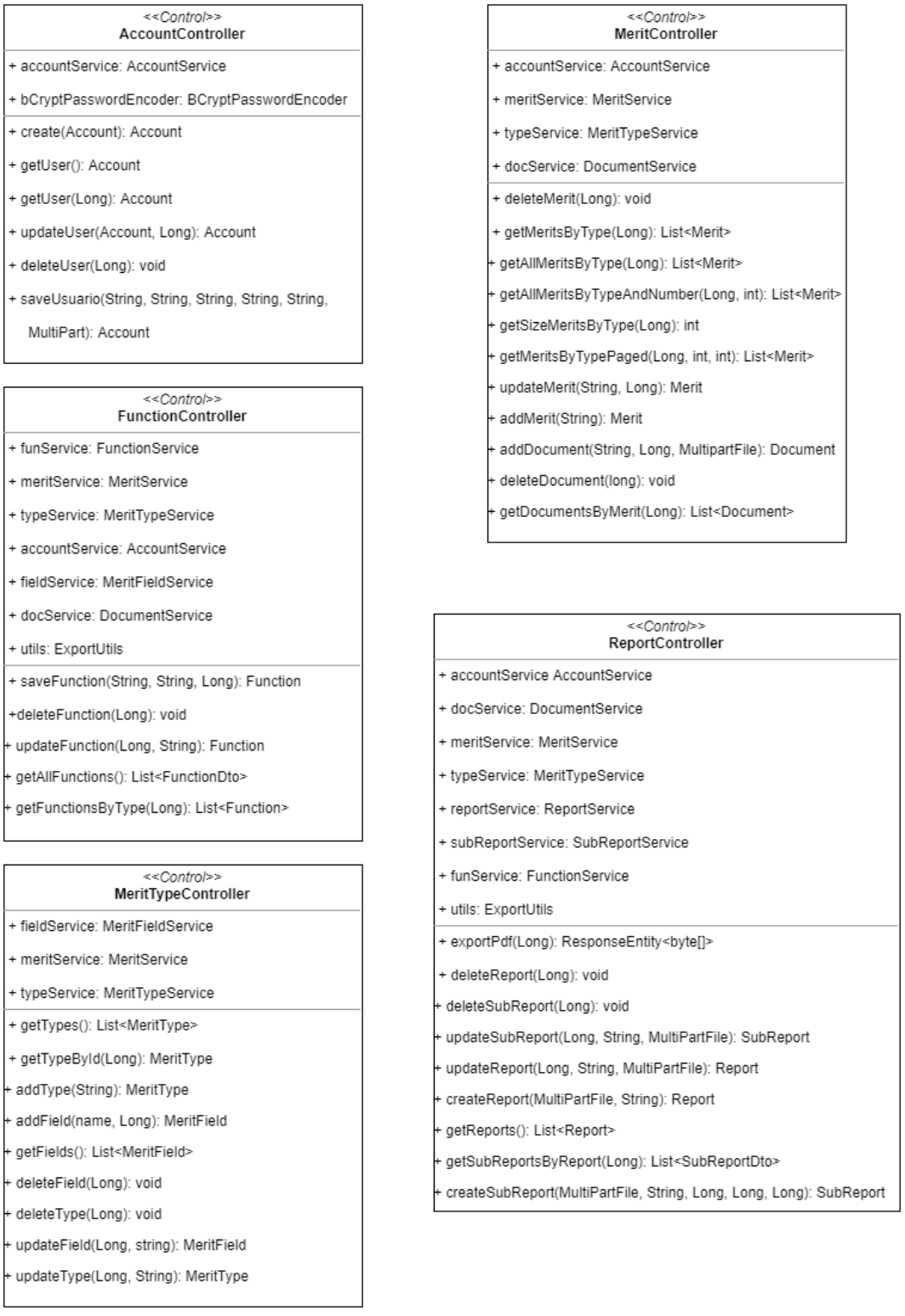

Figura 6.7: Diagrama de controladores del servidor

6.3.4 Interfaz REST

Como ya hemos indicado en la Sección 6.3, el servidor expone las funcionalidades a través de una interfaz REST. En la Figura 6.7 hemos descrito cada uno de los controladores así como sus dependencias y relaciones. En este apartado describiremos con mayor detalle cuales son las operaciones que cada controlador expone, así como las direcciones y métodos necesarios para poder invocar a cada una de estas funcionalidades. Estas operaciones podrán ser utilizadas por cualquier interfaz o aplicación que genere peticiones HTTP a dichas direcciones.

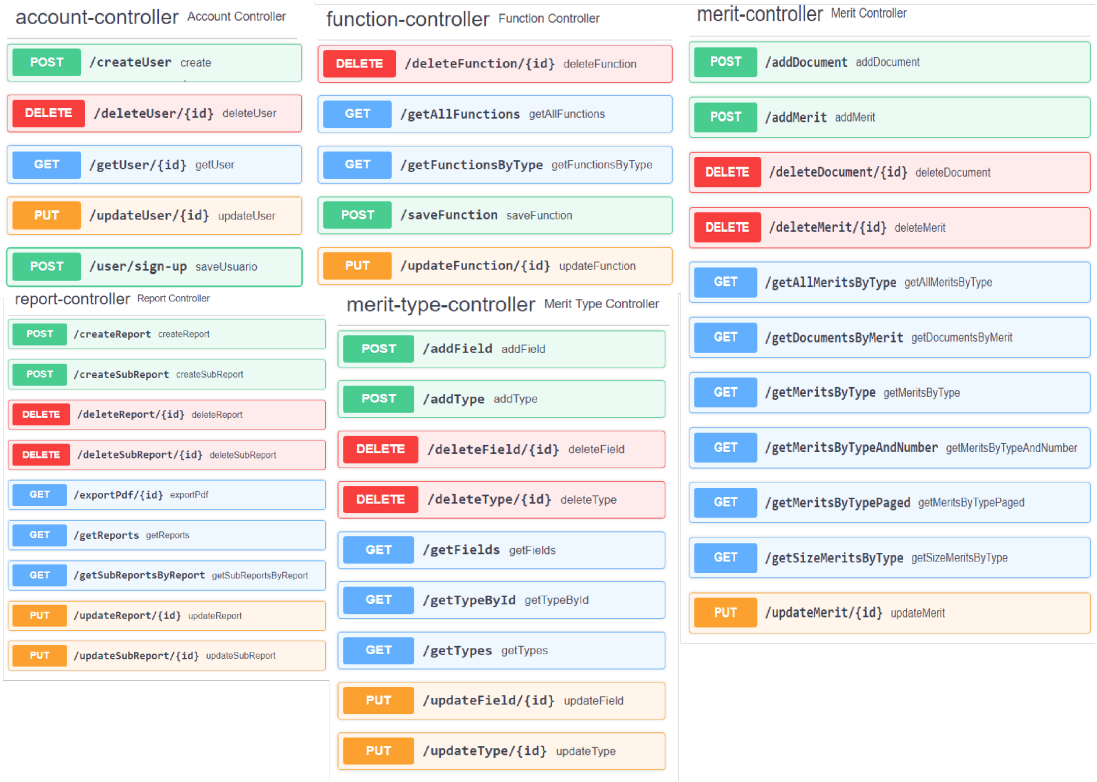

Figura 6.8: Operaciones del servidor

Capítulo 7

Implementación y pruebas

7.1 Introducción

En este capítulo abordaremos detalles más técnicos de la implementación y las pruebas del código que conforman el proyecto. A pesar de que muchos de estos detalles ya han sido ilustrados brevemente, profundizaremos más en los mismos, debido a la importancia que presenta su codificación para el conjunto del sistema.

7.2 Implementación

Tras haber comentado los detalles más básicos de la codificación del proyecto y conocer las funcionalidades ofertadas, ahora conoceremos los algoritmos y la lógica que las hacen posibles. Para esto, profundizaremos en las dos funcionalidades que conforman los pilares del sistema que, además, son las operaciones más complejas e interesantes del proyecto.

7.2.1 Exportación de currículums

Esta es, sin duda alguna, la funcionalidad más importante del sistema, ya que es la encargada de recopilar todos los datos y crear el resultado último de la aplicación, el currículum. Como hemos comentado anteriormente, este método está diseñado para soportar la parametrización y escalabilidad del sistema. Esto quiere decir que el mismo método será usado para generar cualquier formato de currículum, tanto ofertado en el presente, como algún otro que se pudiera añadir en el futuro.

Para lograr esto, hemos diseñado un método dinámico que trabaja con ficheros fuente que representan secciones del currículum y los rellena y compila en función del formato seleccionado y sus necesidades. Además, comprueba antes de rellenar cada sección si los datos presentan peculiaridades como funciones transformadoras aplicadas o documentos adjuntos

y actúan acordemente para reflejarlo en el currículum final. En el siguiente apartado, profundizaremos en cómo se construyen las plantillas, y cómo se utilizan para generar el currículum final. Para ilustrar este proceso de una manera más gráfica, nos ayudaremos del diagrama de actividad del proceso de exportado.

Como veremos en la Figura 7.1, el exportado de currículums es un proceso complejo y dinámico, que se desarrolla en varias etapas:

- Primera etapa: Se recuperan los datos del usuario y la plantilla principal del currículum. Esta plantilla a su vez se compila.
- Segunda etapa: Se recuperan todas las secciones que forman en este momento el currículum. Tras esto, para cada sección, se obtiene la lista de méritos del usuario del tipo que rellene dicha sección. Para cada mérito a su vez, se recuperan todos los documentos adjuntos y se añaden a un archivo que englobará el conjunto de todos los documentos. Además, se comprueba si sobre la sección hay aplicada alguna función y, de ser así, se aplica a los méritos.
- Tercera etapa: Se construyen para cada sección una fuente de datos a partir de los méritos y, posteriormente, se añaden tanto la plantilla de la sección como la fuente de datos a los parámetros de la plantilla principal.
- Cuarta etapa: Por último, se obtiene el fichero PDF resultado de aplicar las secciones rellenadas a la plantilla principal. A este fichero, se le añade el archivo compuesto por todos los documentos adjuntos a los méritos ordenados por aparición. Este único PDF resultante, que será un currículum completo, se devuelve al usuario.

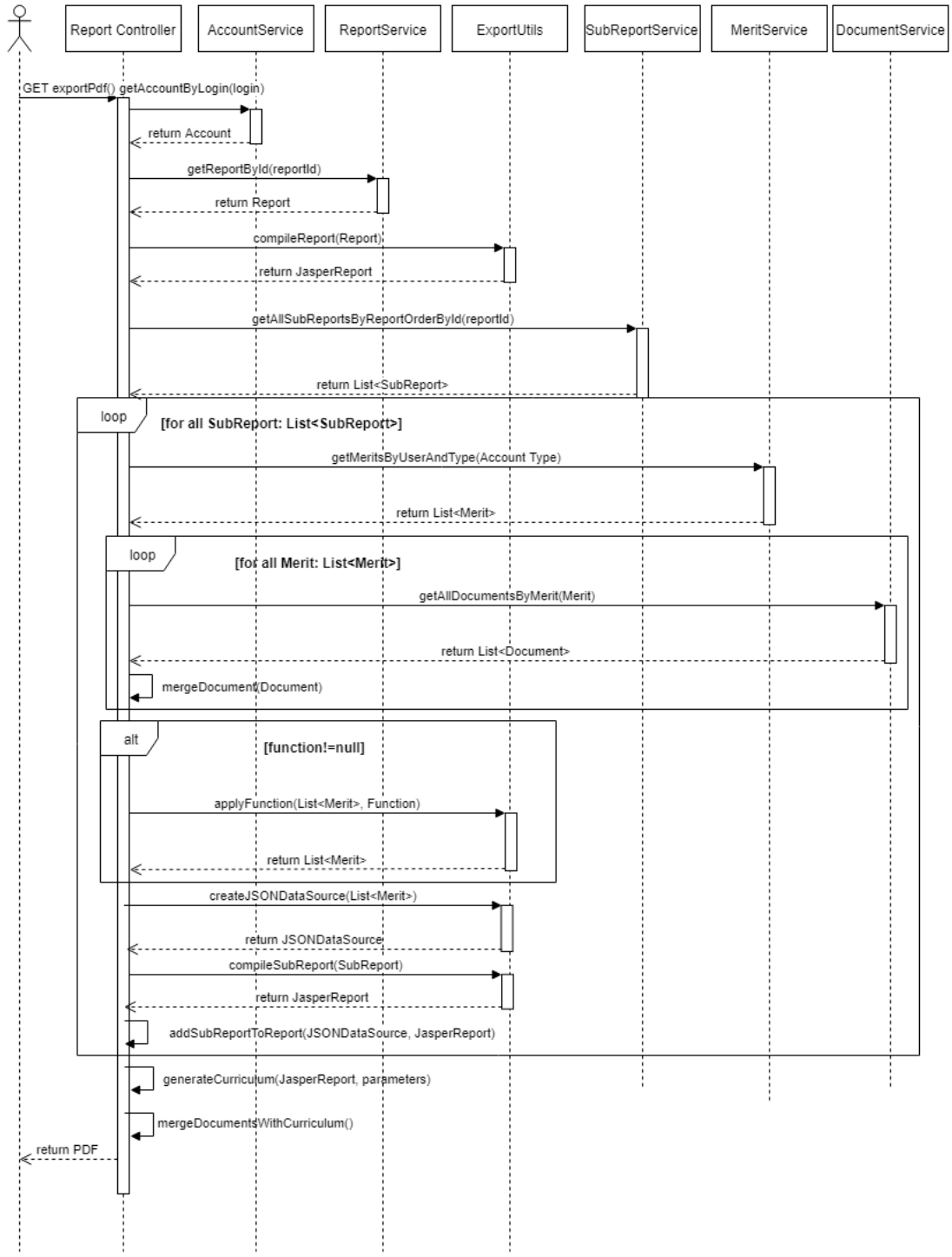

Figura 7.1: Diagrama de secuencia simplificado del proceso de exportado de un currículum

7.2.2 Plantillas JasperReports

Para que el exportado de currículums que acabamos de exponer funcione, hemos empleado como elementos principales plantillas compatibles con la librería Java JasperReports. Para que cada currículum sea funcional debe presentar como mínimo una plantilla principal y una plantilla por cada sección que emplee tipos de datos diferentes. Estos archivos deben ser incluidos en la aplicación por un administrador, asociándolos a su vez a un tipo de dato. Esta asociación será la que permita en el exportado conocer qué méritos serán incluidos en cada sección.

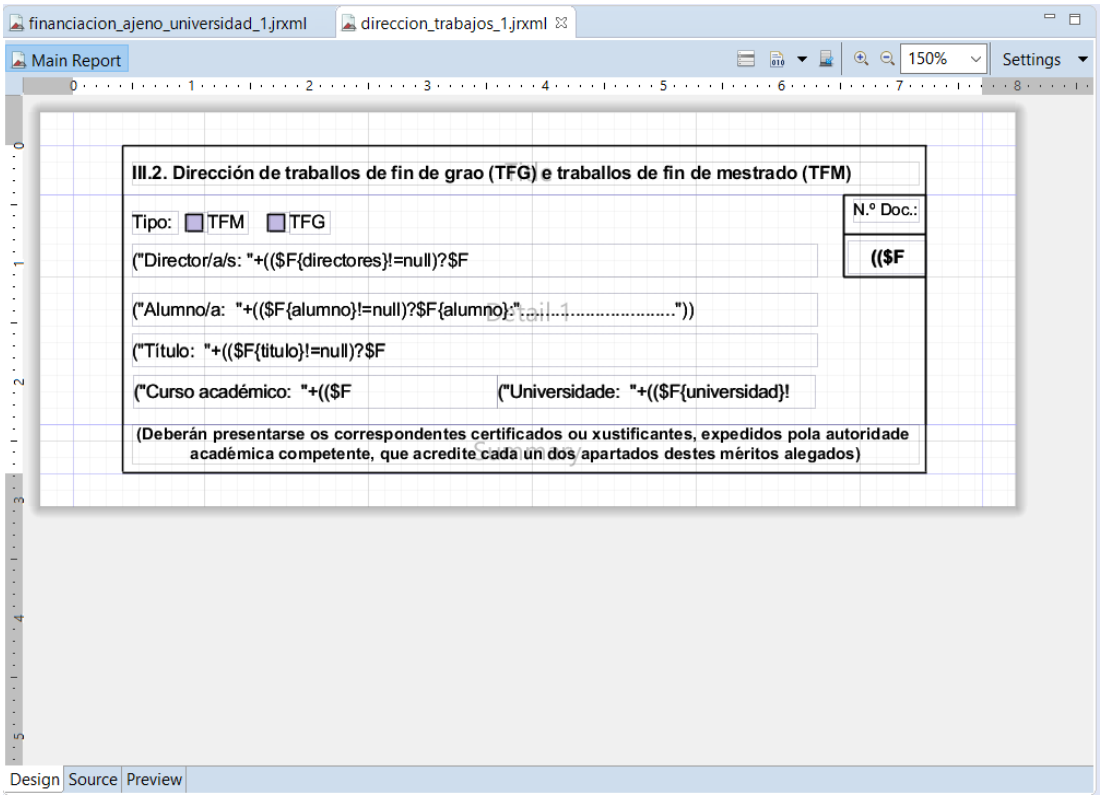

Figura 7.2: Diseño de una sección con JasperSoft Studio

Estas plantillas a las que nos referimos son archivos XML que hemos elaborado ayudándonos de la aplicación JasperSoft, que ya hemos descrito en el Capítulo 2.2. El entorno ofrece un abanico de componentes que podemos emplear en la construcción de la plantilla, simplemente arrastrándolos a nuestro espacio de trabajo.

A la hora de construir plantillas, se usarán como componentes principales etiquetas de texto estáticas para los títulos y etiquetas dinámicas que serán rellenadas con los datos obtenidos del usuario. Estos componentes, junto con otros elementos estéticos permitirán la creación de las plantillas.

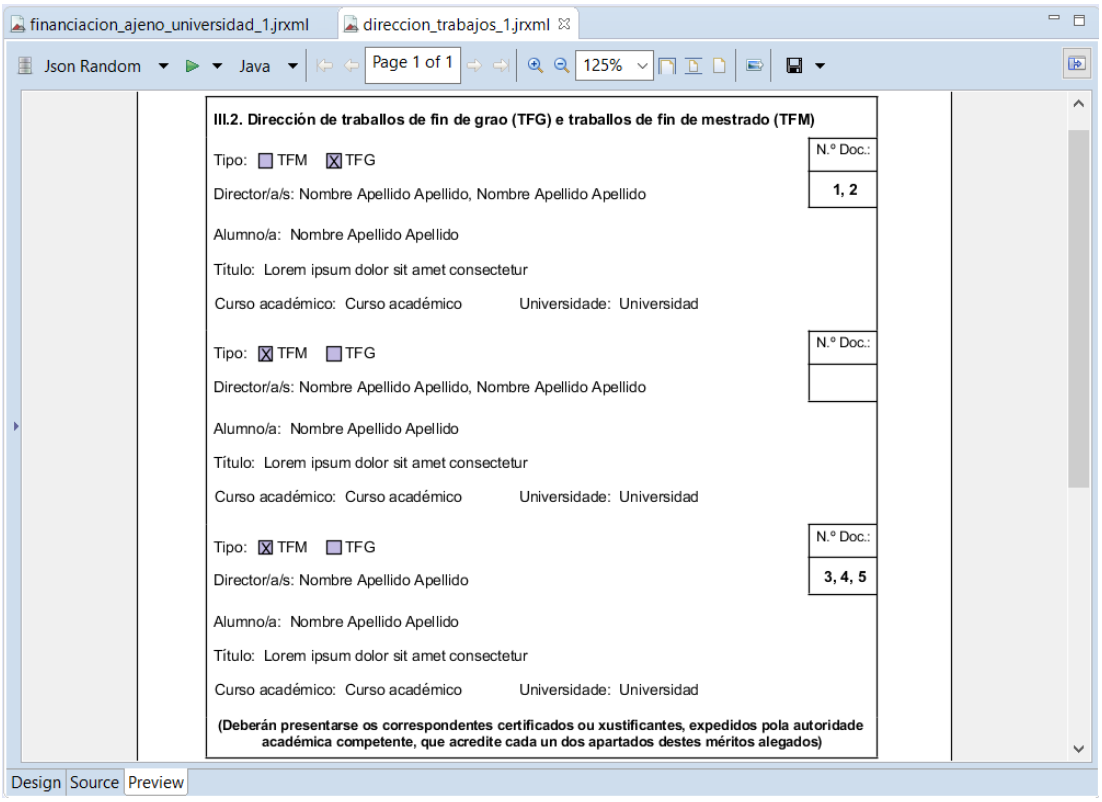

Figura 7.3: Ejemplo de rellenado de una sección con datos de prueba

Como hemos podemos comprobar en la Figura 7.2, todos los campos que forman parte de una sección están compuestos de parámetros que posteriormente serán rellenados por datos, como podemos ver en la Figura 7.3. Para esto, emplearemos unas fuentes de datos JSON, construidas a partir de los méritos del usuario, que automáticamente son mapeadas al parámetro concreto que deben rellenar. Por lo tanto, las plantillas han sido diseñadas para ser escalables y no limitar al usuario la extensión de sus datos o el número de los mismos. Como única restricción, los campos que se rellenarán dinámicamente deben tener el mismo nombre que el campo JSON con el que se van a relacionar.

Para facilitar posteriormente la creación de los tipos de méritos que rellenarán las plantillas, a la hora de crear un currículum nuevo se permite añadir un documento de texto que ayude al administrador a respetar la convención de nombrado que permitirá la correcta construcción final del currículum.

7.2.3 Creador dinámico de méritos

Para que la exportación de currículums pueda tener lugar, primero, el usuario deberá introducir y gestionar toda la información necesaria acerca de sus méritos. Para esto, la aplica-

ción proporciona un creador de méritos global que permite al usuario la creación de cualquier tipo de mérito. Para crear un mérito de esta manera, el usuario tan solo deberá seleccionar el tipo de mérito a crear y, dinámicamente, la aplicación le proporciona los campos que debe cubrir para lograr su objetivo. Este creador proporciona siempre soporte a la creación de cualquier tipo de mérito presente o futuro.

Cada tipo de mérito anteriormente descrito está compuesto por un conjunto de campos, que serán los que posteriormente el usuario deba cumplimentar. Cada tipo de mérito será creado por un administrador que deberá determinar cuales son los campos que lo compondrán. Cuando un usuario quiera añadir un nuevo mérito de un tipo determinado, la aplicación dinámicamente presentará al usuario un formulario compuesto por todos los campos que componen dicho mérito. Esto es posible ya que cada tipo de mérito tiene una relación con todos los campos que lo componen. Todas estas relaciones y estructuras estarán almacenadas persistentemente en una base de datos.

Ante esta generación, vemos que cada mérito perteneciente a tipos diferentes debería presentar atributos diferentes. Esto podría provocar la necesidad de modelar una entidad diferente cada vez que se añadiera un tipo de mérito nuevo a la aplicación. Sin embargo, para solventar el problema, se ha modelado la entidad mérito para que almacene la información introducida por el usuario como una estructura JSON y almacene además el tipo de mérito que representa. Esto nos permite utilizar la misma estructura para almacenar la información para cualquier tipo de mérito que se cree en la aplicación. Posteriormente, para poder extraer la información del JSON de manera correcta, únicamente debemos comprobar de que tipo es el mérito para poder relacionar cada campo JSON con un campo del tipo de mérito concreto.

7.2.4 Creación y aplicación de funciones personalizadas

En estas dos secciones anteriores, hemos comprobado cómo se crean los dos elementos imprescindibles para la creación de los currículums: las plantillas XML y las fuentes de datos que las rellenan. Gracias a estos dos elementos, podemos crear cualquier tipo de currículum pero con un inconveniente principal. A la hora de añadir un currículum nuevo, es posible que una de las secciones precise de un tipo de dato similar a uno ya existente, pero que este no se adecúe completamente a sus necesidades. Como solución al problema, para que el currículum pueda ser creado, deberíamos crear un nuevo tipo de dato que el usuario deberá rellenar, a pesar de que muchos de estos datos ya los ha rellenado con anterioridad. Para solventar esta problemática, la aplicación ofrece la oportunidad de crear funciones JavaScript personalizadas, que permitan transformar los méritos ya existentes en otros nuevos, de la manera que se crea conveniente.

Este editor de funciones es una funcionalidad solo disponible para el administrador, que permitirá a los usuarios evitar la pérdida de tiempo que supondría volver a introducir los mis-

mos datos, mediante la reutilización de tipos ya existentes con pequeñas o grandes modificaciones. Esta herramienta permite aplicar una función JavaScript determinada, que transforme los méritos de la manera deseada.

La entrada de la función JavaScript será un array de méritos, donde cada mérito presenta la misma estructura JSON. La salida de la función a su vez, deberá devolver también un array de méritos con la misma estructura JSON entre ellos pero transformada. Como ejemplo de esta funcionalidad podemos observar una transformación de datos en la Figura 7.5. Aquí, se ha modificado la estructura del tipo de mérito transformándolo en otro con unos campos diferentes, pero devolviendo igualmente un array de méritos.Además, el editor permitirá al creador de la función observar cual será el resultado de la aplicación de dicha función sobre unos méritos de ejemplo.

Una vez el resultado sea el satisfactorio, la función JavaScript será persistida en base de datos y será relacionada con el tipo de mérito que transforma. Posteriormente, a la hora de añadir subplantillas a un currículum, el usuario podrá decidir si aplicar una función de las que ya han sido creadas sobre el tipo de mérito seleccionado. La aplicación de dicha función para transformar los méritos se realiza en el servidor y será empleada durante la exportación del currículum, cuando este presente una subplantilla con una función aplicada.

Sumado a una transformación habitual, existe la posibilidad de que en lugar de guardar la función, guardemos el resultado de su aplicación como nuevo tipo de mérito, que será utilizable de la manera habitual.

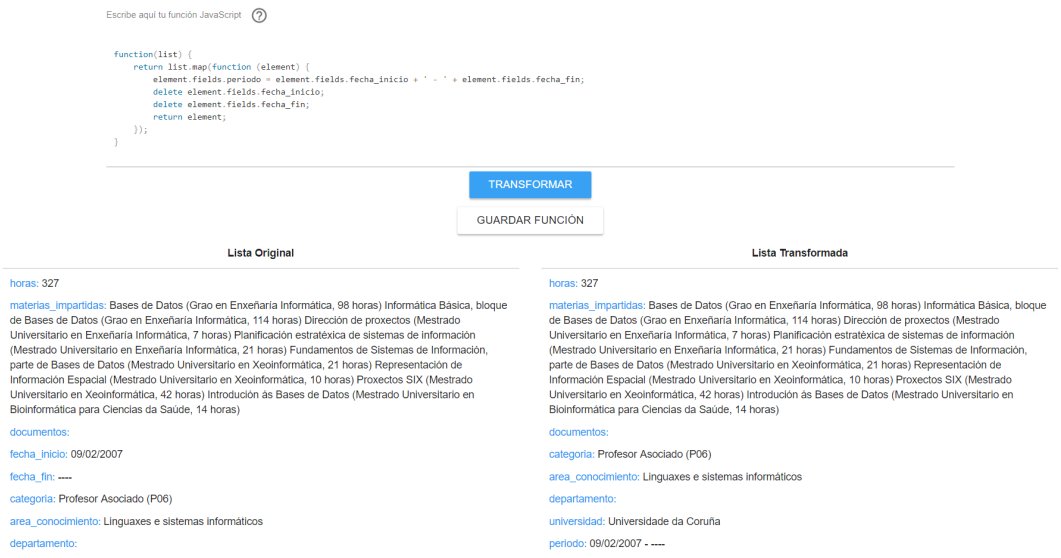

Figura 7.4: Ejemplo de creación de funciones personalizadas

7.3 Pruebas

Para comprobar el correcto funcionamiento del sistema, se han diseñado, para cada funcionalidad ofertada por el servidor, un conjunto de pruebas de integración a través de la herramienta SOAP-UI. Al tratarse de pruebas realizadas contra un controlador REST, las pruebas se corresponden con peticiones web sobre las direcciones a probar, donde se comprueba tanto que el tipo devuelto sea el deseado como que también lo sean las cabeceras y estados de las respuestas. Este conjunto englobará siempre dos tipos de pruebas:

- **Pruebas correctas**: Utilizamos este tipo de pruebas, para comprobar el correcto proceder de una funcionalidad, para una llamada estándar. De esta prueba esperamos que la respuesta devuelva un estado correcto, así como el valor deseado.
- **Pruebas incorrectas**: Utilizamos este tipo de pruebas para comprobar cómo responde nuestra aplicación ante todo tipo de errores del usuario. Dentro de este tipo de pruebas, incluiremos errores en el número o valor de los parámetros o circunstancias especiales que puedan provocar situaciones no deseadas. De este tipo de prueba esperamos que devuelva un estado de error concreto y, quizá, algún tipo de información extra.

Además, estas pruebas se pueden estructurar en conjuntos de peticiones que permiten comprobar el funcionamiento del servidor para un conjunto de operaciones consecutivas. Podemos usar estas pruebas para comprobar si el servidor opera de manera eficiente.

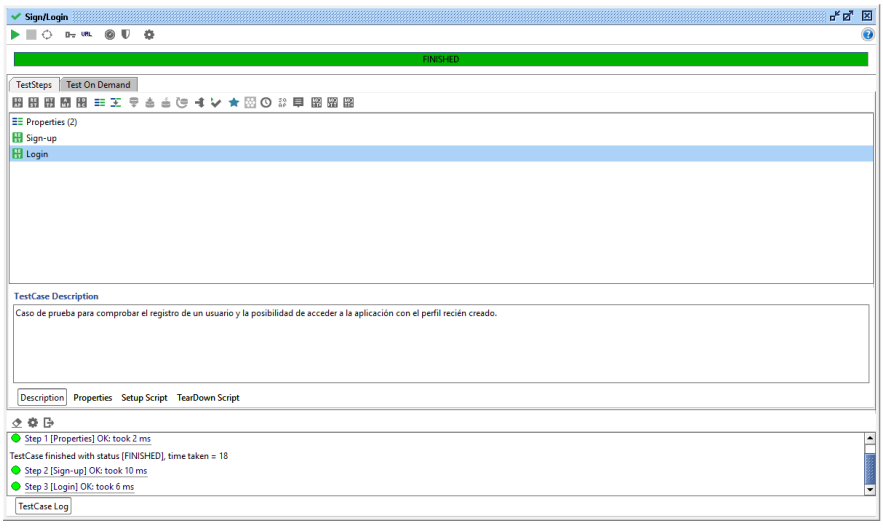

Figura 7.5: Caso de prueba para comprobar el registro y login del usuario

Capítulo 8

Solución desarrollada

8.1 Introducción

En este capítulo, vamos a navegar entre las principales funcionalidades ofertadas por la aplicación, acompañando las descripciones con imágenes del sistema final.

8.2 Ingreso en el sistema

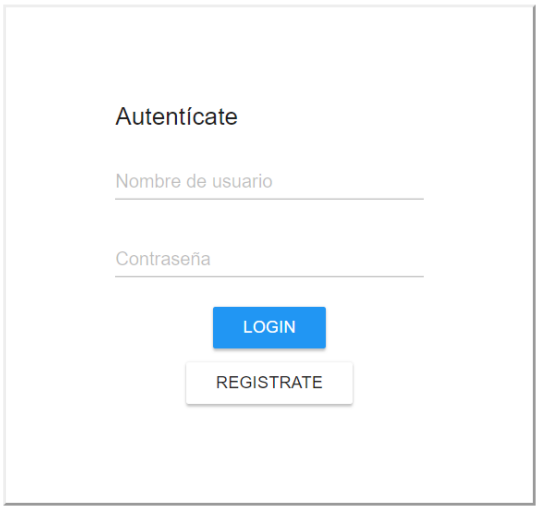

Figura 8.1: Página de ingreso a la aplicación

Para poder acceder a todas las funcionalidades del sistema, el usuario debe ingresar en la aplicación a través del uso de un nombre de usuario y una contraseña que debe haber creado previamente. Este inicio de la aplicación es compartido tanto por usuarios finales como por los administradores, decidiendo a qué vista redirigirlos en función de los privilegios del usuario.

Además, desde aquí se puede acceder a un formulario de registro, en caso de que todavía no poseas una cuenta creada.

8.3 Vista del usuario

En esta sección, analizaremos las funcionalidades y las vistas que posee un usuario común dentro de la aplicación.

8.3.1 Gestión de méritos

Mediante este concepto englobamos la mayoría de las funcionalidades expuestas para los usuarios comunes. La mayoría de las operaciones consisten en la visualización, inclusión, actualización y borrado de méritos.

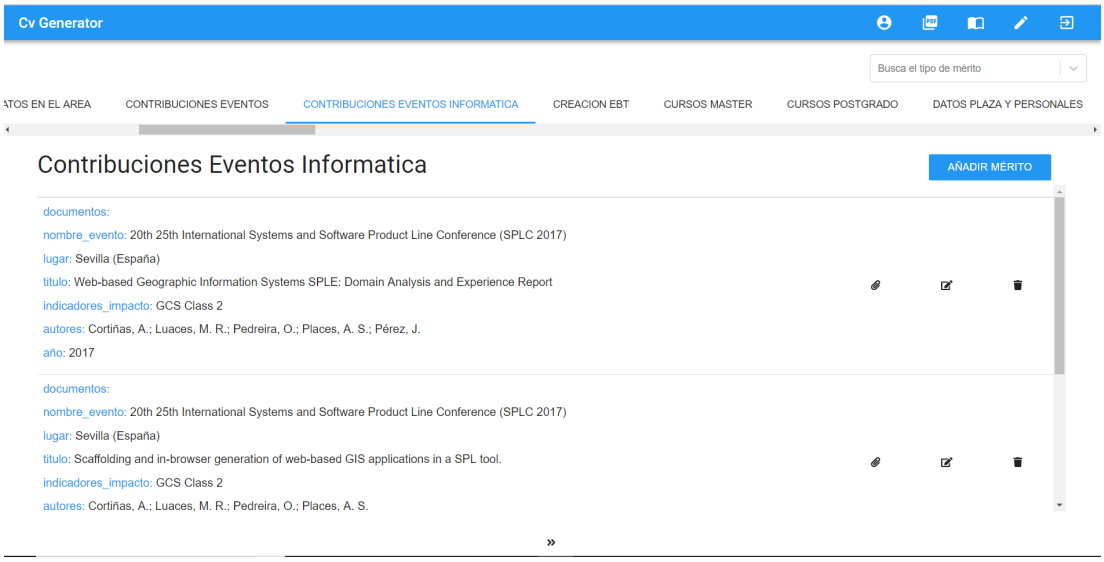

Figura 8.2: Vista principal del usuario

Desde la vista principal, el usuario puede navegar entre todos los tipos de méritos existentes, a través de un tabulador o buscándolos por su nombre. Una vez seleccionado un tipo de mérito, puede visualizar todos los méritos de ese tipo que tiene creados, realizar cualquier operación con ellos o incluir uno nuevo.

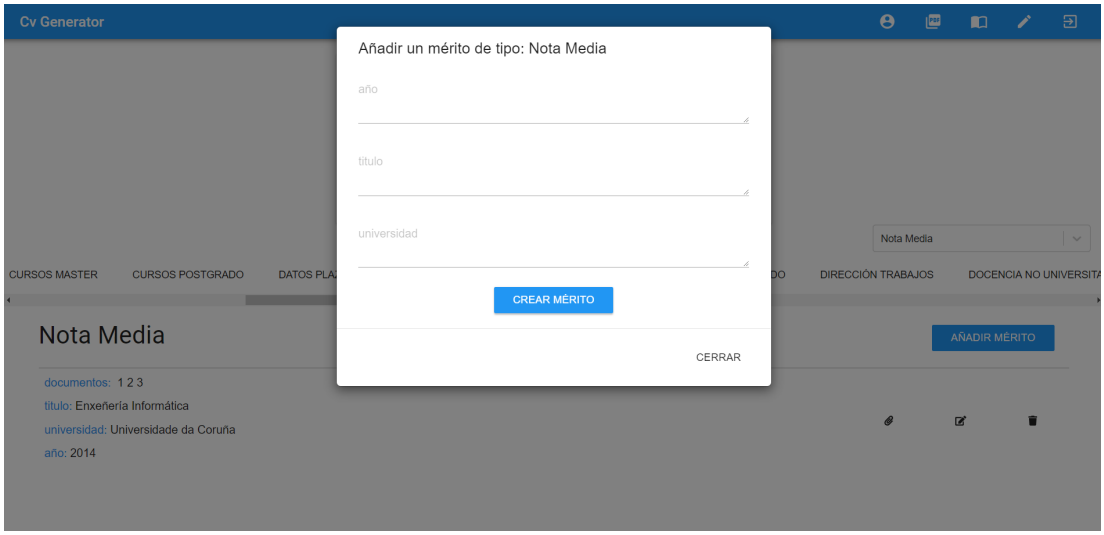

Figura 8.3: Diálogo de creación de un nuevo mérito

Cualquiera de estas funcionalidades desplegará en la pantalla un diálogo, que expondrá lo necesario para completar dicha operación. Como podemos ver en la Figura 8.3, añadir un mérito será una operación tan sencilla como cumplimentar los campos que deseemos de entre los que conforman dicho tipo. Diálogos similares serán los que se desplieguen tanto en el resto de operaciones básicas como a la hora de anexionar documentos a un mérito, como podemos ver en la Figura 8.4.

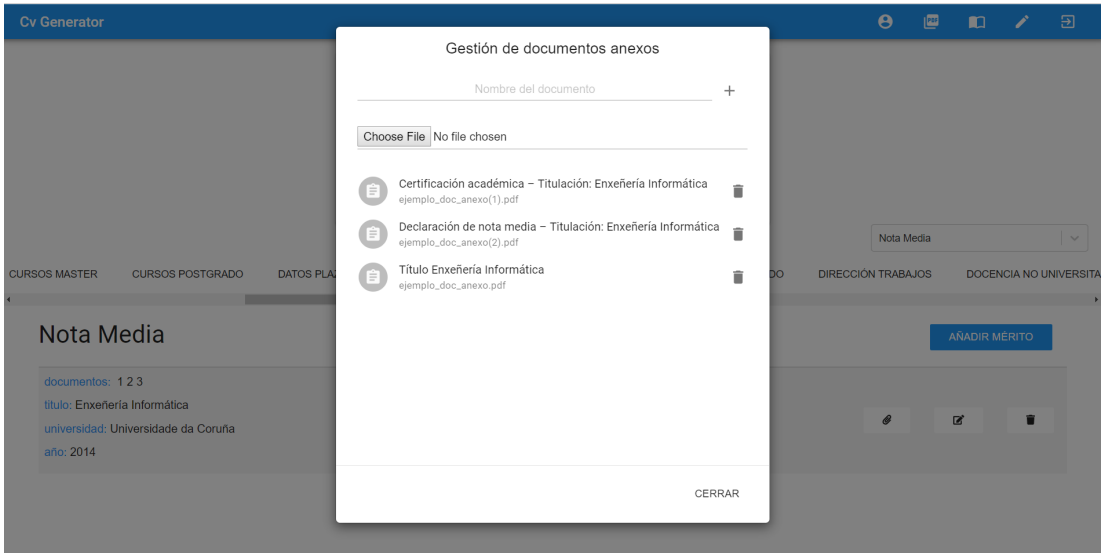

Figura 8.4: Diálogo de anexión de documentos al mérito

8.3.2 Barra de navegación

El usuario cuenta con una barra de navegación, como hemos visto en la figura 8.2, que nos expone todas las funcionalidades que no están ligadas directamente a un tipo de mérito concreto. Estas funcionalidades son:

- Creación global de méritos: Un creador que nos permite añadir un mérito de cualquier tipo únicamente seleccionando previamente el tipo a crear.
- Desconexión de la aplicación: Cierra la sesión del usuario en la aplicación.
- Visualización del perfil de usuario: Permite al usuario visualizar su perfil y acceder a la edición del mismo.
- Exportación de currículums: Permite al usuario exportar el currículum, simplemente seleccionando el formato deseado de entre todos los ofertados.
- Visualización de manuales de ayuda a la creación: Esto permite al usuario acceder al manual de ayuda a la creación para cualquier formato de currículum ofertado, simplemente seleccionándolo de la lista.

8.3.3 Creación de currículums

La otra gran funcionalidad para los usuarios es la creación de currículums, a través de los méritos que han creado aprovechándose de las operaciones que acabamos de mostrar. Para realizar dicha función, el usuario primero puede consultar o descargarse el manual de ayuda de cualquier formato. Esta operación y la de exportar un currículum serán igualmente fáciles, ya que basta con seleccionar el formato entre todos los disponibles. La aplicación, a partir de este punto, realizará todo lo necesario para proporcionar el currículum final al usuario, sin necesitar mayor intervención por parte del mismo.

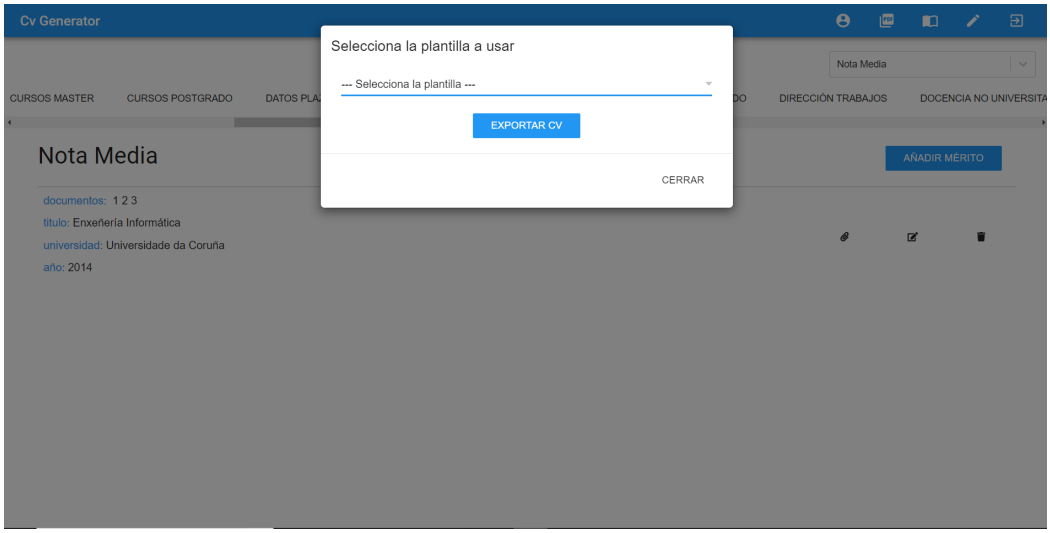

Figura 8.5: Selección del formato para exportar el currículum

8.4 Vista del administrador

En esta sección, analizaremos las funcionalidades y las vistas que posee un usuario con privilegios de administrador. Antes de profundizar en los grandes grupos de funcionalidades, cabe destacar que, en este caso, la barra de navegación no desplegará funcionalidades, sino que nos permite navegar entre las tres vistas que comentaremos a continuación.

8.4.1 Gestión de tipos de méritos

Una vez ingresamos en la aplicación como administrador, nos encontramos con el primero de los tres grandes bloques de funcionalidades, la gestión de los tipos de méritos.

Englobamos dentro de estas funcionalidades la inclusión, edición, consulta y borrado de los diferentes tipos de méritos, así como los campos que los forman. La navegación entre los tipos es similar a la realizada por los usuarios, pudiendo usar los selectores o realizar una búsqueda por nombre. Una vez seleccionado el tipo, existe un desplegable con todas las operaciones que podemos realizar sobre dicho tipo. Seleccionada la operación deseada, se desplegará un diálogo, similar a todos los ya vistos, que permite acometer la acción.

Figura 8.6: Vista de la gestión de tipos de mérito

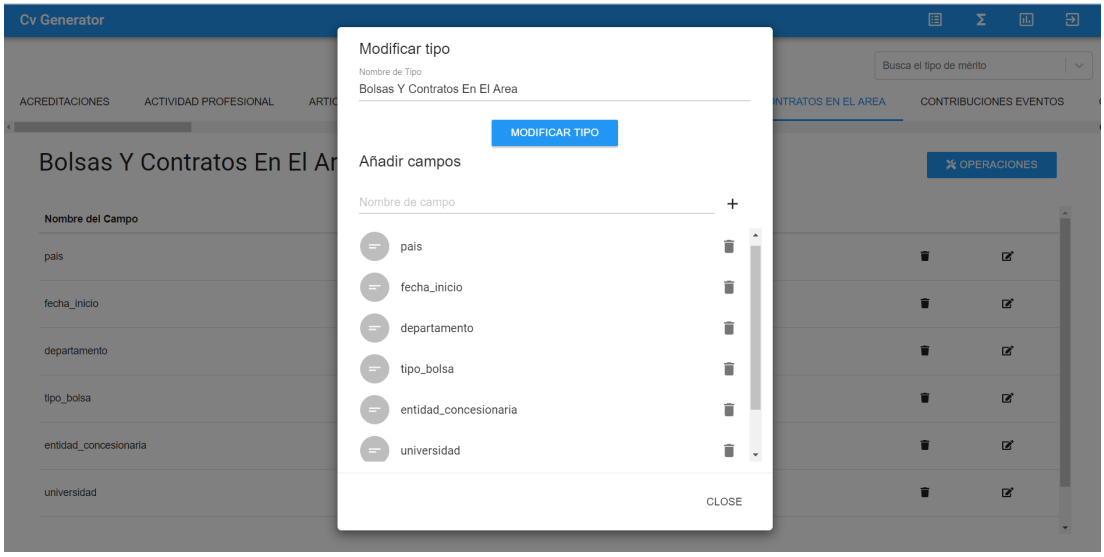

Figura 8.7: Diálogo de edición de un tipo de mérito

8.4.2 Gestión de plantillas

Se trata de una vista muy similar a la anterior, con la salvedad de que en este caso gestionamos las plantillas que corresponden a los diferentes currículums y sus secciones. La vista y navegación son completamente similares, así como los diálogos desplegados tras seleccionar cualquier operación, como podemos ver en la figura 8.9.

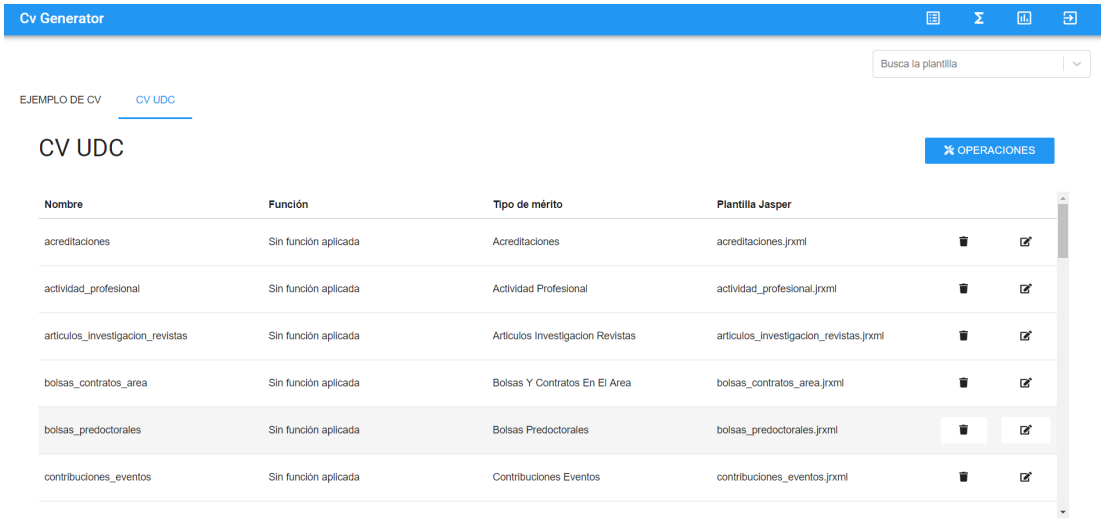

Figura 8.8: Vista de la gestión de plantillas

8.4.3 Gestión de funciones

Este bloque presenta la vista más diferente de las tres tanto visual como funcionalmente. Esta vista nos permite la gestión de las funciones que se aplican a los tipos de méritos. Posee dos espacios diferenciados: un editor de funciones como ya hemos vista en la figura 7.5 y un visor de las funciones ya creadas.

A pesar de que se trata de una vista que proporciona una funcionalidad muy importante, la interfaz es sencilla y amigable, para ayudar en la consecución de la tarea. Además, se incluye un diálogo de ayuda a la creación de funciones, así como un ejemplo, para permitir al administrador el uso correcto y fácil del editor.

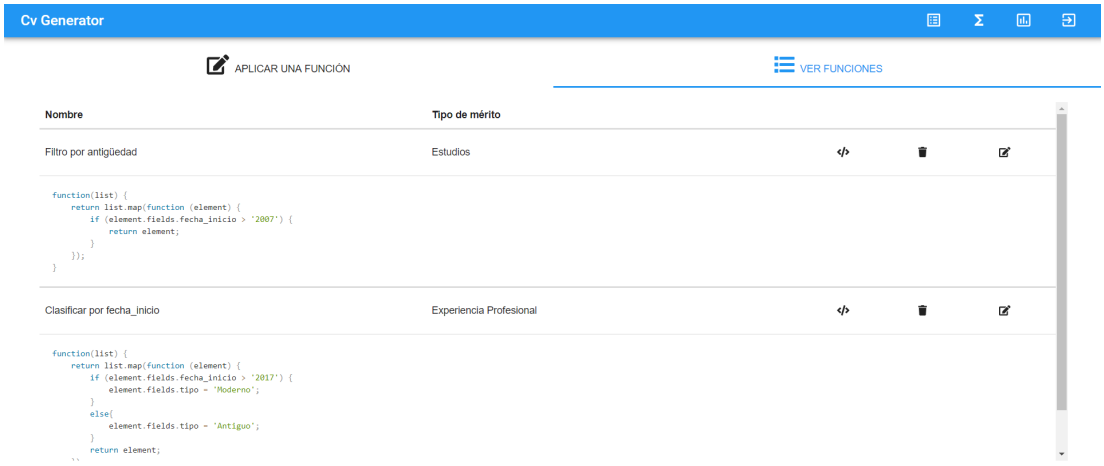

Figura 8.9: Vista de funciones guardadas

Capítulo 9

Conclusiones

9.1 Conclusiones

El proyecto ha permitido el desarrollo de un sistema de gestión de méritos y exportación de currículums funcional y completo. Durante el desarrollo, se han podido abordar todas las funcionalidades que en un principio se habían planificado, así como algunos requisitos funcionales extra que surgieron posteriormente. Esto ha sido posible, en gran parte, gracias al empleo de una metodología ágil, como hemos explicado en el capítulo 3. La creación de sistemas completos y funcionales cada poco tiempo permite que la aplicación evolucione de una manera más fácil, siendo sencillo la adición de nuevas funcionalidades o la modificación de tareas durante el proceso del desarrollo. Esto tiene como resultado final un sistema muy ajustado a las expectativas y necesidades de la actualidad.

Como hemos analizado durante el capítulo 2, la aplicación presenta similitudes con aplicaciones existentes en el mercado, pero mejora sus prestaciones, destacando entre ellas la escalabilidad y la adaptación. Esto permite que la aplicación pueda crear cualquier tipo de currrículum actual y futuro sin la necesidad de modificación alguna del código actual. Además, se trata de una herramienta única que puede englobar multitud de formatos, permitiendo a los usuarios gestionar todos sus méritos y currículums desde un único entorno centralizado.

El desarrollo ha permitido profundizar en el uso de tecnologías punteras en la actualidad del desarrollo:

Metodología ágil : Se ha empleado la adaptación de una metodología ágil de desarrollo, SCRUM, que permite evitar los grandes problemas de las metodologías clásicas y hace del desarrollo del sistema un proceso que evoluciona con el tiempo.

Cliente SPA : Se ha diseñado una aplicación cliente SPA que permite solucionar algunos problemas de las aplicaciones basadas en un conjunto de páginas. Además, esta arquitectura de diseño es más eficiente en cuanto al rendimiento se refiere.

Framework de desarrollo web : Para la codificación de la aplicación cliente se ha empleado uno de los grandes framework de desarrollo web, como es React. El desarrollo del sistema ha permitido su aprendizaje desde cero y, conforme avanzaba el proyecto, la profundización en el lenguaje.

Arquitectura : Se ha trabajado con una arquitectura en capas una de las más empleadas en la actualidad, donde todos los componentes que la forman son totalmente independientes.

Plantillas : El desarrollo del proyecto ha permitido profundizar en el conocimiento de los diferentes formatos del currículum, así como en el diseño de plantillas a través de la librería JasperReports.

9.2 Trabajo futuro

Si bien la aplicación es completa conforme a los requisitos iniciales, has surgido un conjunto de posibles líneas de trabajo futuro.

Una funcionalidad interesante sería la posibilidad de importar los méritos de los usuarios a través de portales desde donde dichos datos sean accesibles. Ejemplos de estos portales web podrían ser Google Scholar [24] o Research Gate [25]. Esto permitiría evitar al usuario la introducción manual de algunos tipos de méritos, como publicaciones realizadas o congresos asistidos.

Por último, sería interesante contar con un editor de plantillas de forma que no hiciera falta utilizar ninguna aplicación externa. Como se ha indicado en la sección 7.2.2, el proceso de creación de plantillas se realiza mediante la aplicación JasperSoft Studio y se debe crear un archivo independiente para cada una de las secciones de un currículum. La posibilidad de crear estas plantillas XML automáticamente, indicando únicamente el formato del tipo de mérito que la va a rellenar, permitiría que el proceso de añadir formatos nuevos resultara de una simpleza increíble.

Apéndices

Apéndice A

Glosario de acrónimos

CVN *Currículum Vitae Normalizado.*

FECYT *Fundación Española para la Ciencia Y la Tecnología.*

CVA *Currículum Vitae Abreviado.*

API *Application Programming Interface.*

JSON *JavaScript Object Notation.*

JWT *JSON Web Token.*

HTTP *Hypertext Transfer Protocol.*

SGBD *Sistema Gestor de Base de Datos.*

REST *Representational State Transfer*

CRUD *Create Read Update Delete*

SPA *Single Page Application*

AJAX *Asynchronous JavaScript And XML*

XML *EXtensible Markup Language*

SOAP *Simple Object Access Protocol*

UI *User Interface*

HTML *HyperText Markup Language*

SQL *Structured Query Language*

Apéndice B

Glosario de termos

Bibliografía

[1] *Editor de currículums de Europass*, https://europass.cedefop.europa.eu/editors/es/cv/compose.

[2] *Editor de CVN de FECYT*, https://cvn.fecyt.es/editor/.

[3] *Documentación Java*, https://docs.oracle.com/en/java/index.html.

[4] *Página principal de Spring Boot*, https://spring.io/projects/spring-boot.

[5] *Tutorial de Spring Boot y React*, https://spring.io/guides/tutorials/react-and-spring-data-rest/.

[6] *Página principal de Apache PDF Box*, https://pdfbox.apache.org/.

[7] *Página principal de la librería Java JasperReports*, https://community.jaspersoft.com/project/jasperreports-library.

[8] *Tutorial básico de la librería Java JasperReports*, https://www.tutorialspoint.com/jasper_reports/jasper_getting_started.htm.

[9] *Página principal de React*, https://reactjs.org/.

[10] *Tutorial básico de React*, https://reactjs.org/tutorial/tutorial.html.

[11] *Página principal de JavaScript*, https://www.javascript.com/.

[12] *Documentación de BabelJs*, https://babeljs.io/docs/en/.

[13] *Página principal de Material-UI*, https://material-ui.com/.

[14] *Página principal de MUICSS*, https://www.muicss.com/docs/v1/react/introduction.

[15] *Documentación del Estándar Abierto JSON Web Token*, https://jwt.io/.

[16] *Repositorio y tutorial de una implementación Java de JWT*, https://github.com/auth0/java-jwt.

[17] *Página principal de Eclipse*, https://www.eclipse.org/.

[18] *Página principal de Visual Studio Code*, https://code.visualstudio.com/.

[19] *Página principal de SOAP UI*, https://www.soapui.org/.

[20] *Página principal de GitLab*, https://about.gitlab.com/.

[21] *Página principal de JasperSoft Studio*, https://community.jaspersoft.com/project/jaspersoft-studio.

[22] *Guía de las prácticas de la metodología SCRUM*, https://www.scrum.org/resources/what-is-scrum.

[23] *Página principal de Trello*, https://trello.com.

[24] *Página principal de Google Scholars*, https://scholar.google.es/.

[25] *Página principal de Research Gate*, https://www.researchgate.net/.